

msdn magazine



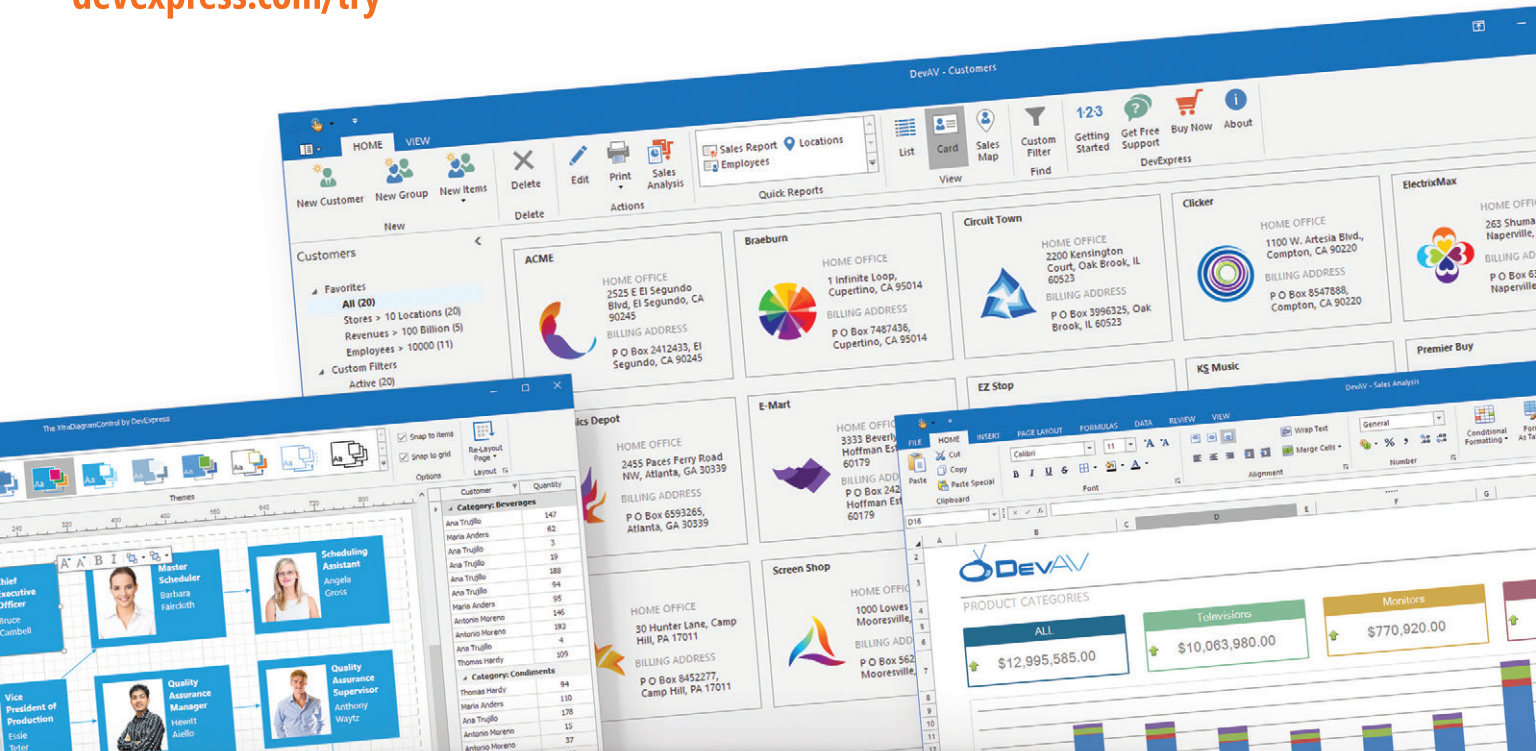
Machine Learning
on the Edge.....15



The King of Desktop Development

Your peers have voted DevExpress Desktop Components best-in-class for 6 straight years. We invite you to see why. Download your free 30-day trial today.

devexpress.com/try





Your Next Great Desktop App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress desktop controls will help you build your best, without limits or compromise.



Download your free 30-day trial
and experience the DevExpress difference today.

devexpress.com/try

msdn

magazine



Machine Learning
on the Edge.....15

Machine Learning with IoT Devices on the Edge James McCaffrey	15
Improving LUIS Intent Classifications Zvi Topol	22
Decentralized Applications with Azure Blockchain as a Service Stefano Tempesta	30

COLUMNS

DATA POINTS

EF Core 2.1 Query Types
Julie Lerman, page 6

THE WORKING PROGRAMMER

How To Be MEAN:
Dynamically Angular
Ted Neward, page 12

CUTTING EDGE

Online Users, Streaming
and Other SignalR Goodies
Dino Esposito, page 40

TEST RUN

Introduction to DNN Image
Classification Using CNTK
James McCaffrey, page 46

DON'T GET ME STARTED

Building Better Meetings
David S. Platt, page 56

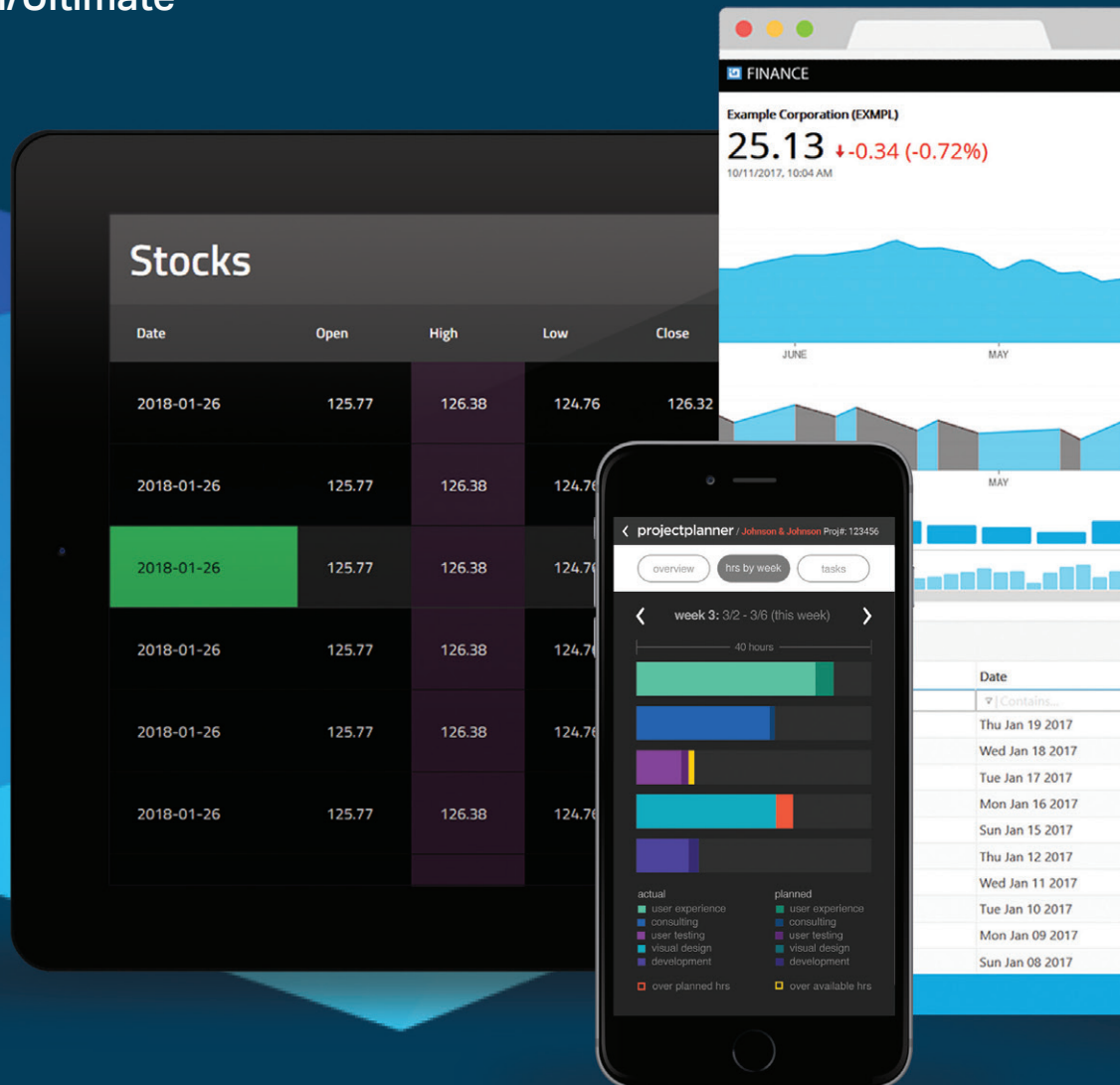


Faster Paths to Amazing Experiences

Infragistics Ultimate includes 100+ beautifully styled, high performance grids, charts & other UI controls, plus productivity tools for building web, desktop and mobile apps.

Angular | JavaScript / HTML5 | ASP.NET | Windows Forms | WPF | Xamarin

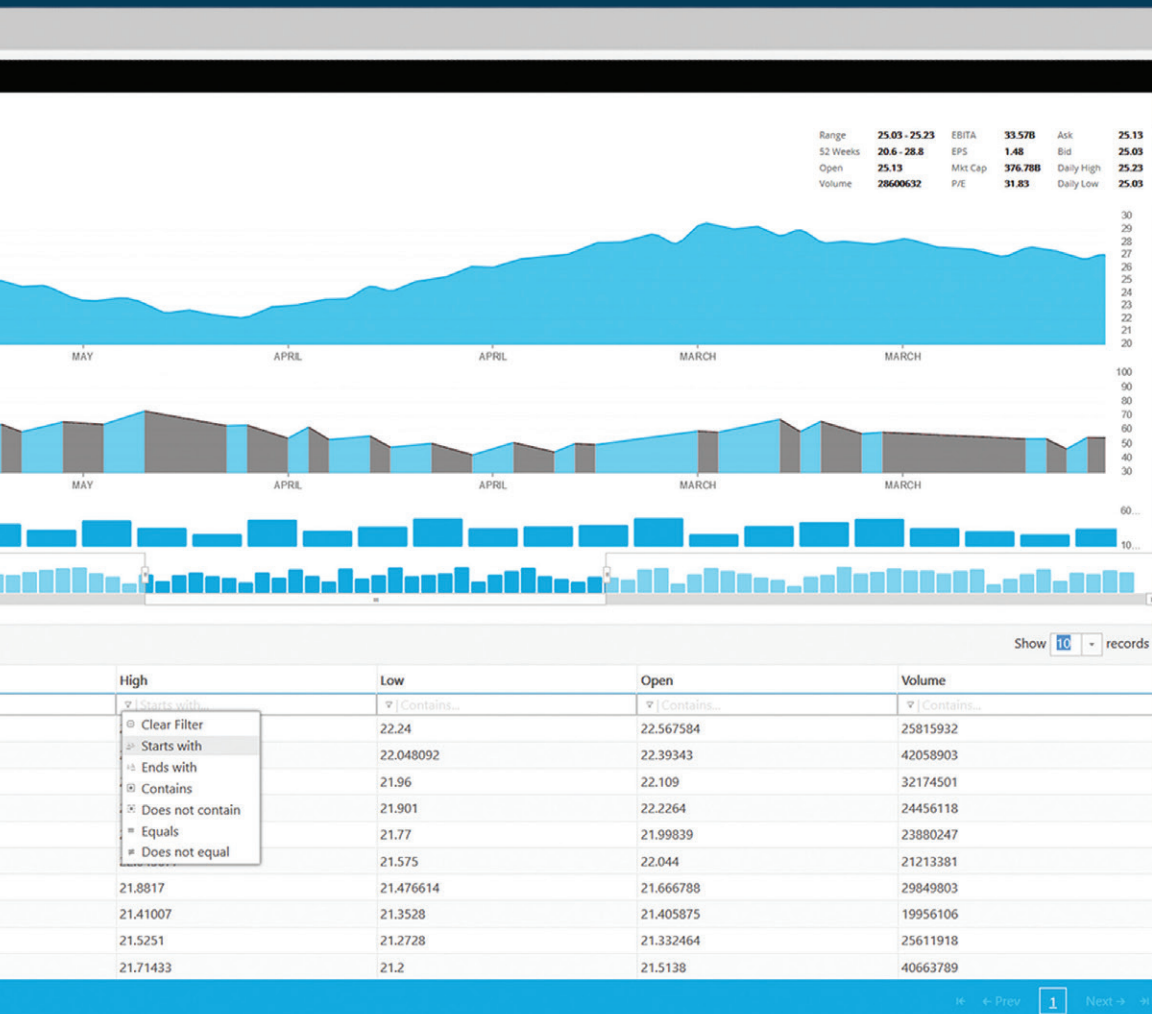
Get started today with a free trial:
Infragistics.com/Ulimate



New Release

Infragistics Ultimate 18.1

- ✓ Fastest **grids & charts** on the market for the Angular developer
- ✓ The most complete **Microsoft Excel & Spreadsheet Solution** for .NET & JavaScript
- ✓ UI controls designed to meet the demands of the toughest **financial & capital market apps**



msdn magazine

JULY 2018 VOLUME 33 NUMBER 7

General Manager Jeff Sandquist

Director Dan Fernandez

Editorial Director Jennifer Mashkowski mmeditor@microsoft.com

Site Manager Kent Sharkey

Editorial Director, Enterprise Computing Group Scott Bekker

Editor in Chief Michael Desmond

Features Editor Sharon Terdeman

Group Managing Editor Wendy Hernandez

Senior Contributing Editor Dr. James McCaffrey

Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt

Vice President, Art and Brand Design Scott Shultz

Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

ART STAFF

Creative Director Jeffrey Langkau
Associate Creative Director Scott Rovin
Art Director Michele Singh
Art Director Chris Main
Senior Graphic Designer Alan Tao
Senior Web Designer Martin Peace

PRODUCTION STAFF

Print Production Manager Peter B. Weller
Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca
Regional Sales Manager Christopher Kourtoglou
Advertising Sales Associate Tanya Egenolf

ONLINE/DIGITAL MEDIA

Vice President, Digital Strategy Becky Nagel
Senior Site Producer, News Kurt Mackie
Senior Site Producer Gladys Rama
Site Producer, News David Ramel
Director, Site Administration Shane Lee
Front-End Developer Anya Smolinski
Junior Front-End Developer Casey Rysavy
Office Manager & Site Assoc. James Bowling

LEAD SERVICES

Vice President, Lead Services Michele Imgrund
Senior Director, Audience Development & Data Procurement Annette Levee
Director, Audience Development & Lead Generation Marketing Irene Fincher
Director, Client Services & Webinar Production Tracy Cook
Director, Lead Generation Marketing Eric Yoshizuru
Senior Program Manager, Client Services & Webinar Production Chris Flack
Project Manager, Lead Generation Marketing Mahal Ramos

ENTERPRISE COMPUTING GROUP EVENTS

Vice President, Events Brent Sutton
Senior Director, Operations Sara Ross
Senior Director, Event Marketing Mallory Bastionell
Senior Manager, Events Danielle Potts
Senior Marketing Coordinator, Events Michelle Cheng



Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 4 Venture, Suite 150, Irvine, CA 92618.

LEGAL DISCLAIMER The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

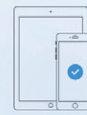
REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
Web: 1105Reprints.com

LIST RENTAL This publication's subscriber list is not available for rental. However, other lists from 1105 Media, Inc. can be rented. For more information, please contact our list manager: Jane Long, Merit Direct
Phone: 913-685-1301;
E-mail: jloug@meritdirect.com;
Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastName@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contents herein do not necessarily express those of the publisher.





WHAT DOES YOUR CODE LOOK LIKE?

```
SEARCH ocrdemo.cs X
```

```
1 using System;
2 using Leadtools;
3 using Leadtools.Ocr;
4 using Leadtools.Document.Writer;
5
6 namespace LEADocrSimpleDemo
7 {
8     class Program
9     {
10         static void Main(string[] args)
11         {
12             //Set the LEADTOOLS evaluation license
13             RasterSupport.SetLicense(@"License\LEADTOOLS.LIC",
14                                     System.IO.File.ReadAllText(@"License\LEADTOOLS.LIC.KEY"));
15             //Specify input and output parameters
16             string inputFile = @"C:\Users\Public\Documents\LEADTOOLS Images\OCR1.TIF";
17             string outputFile = @"C:\Users\Public\Documents\LEADTOOLS Images\OCR1.PDF";
18
19             //Call OCRImage method
20             OCRImage(inputFile, outputFile);
21         }
22
23         static void OCRImage(string inputFile, string outputFile)
24         {
25             Console.WriteLine($"Loading and recognizing {inputFile}");
26
27             //Initiate the LEADTOOLS OCR Engine
28             using (IOcrEngine ocrEngine = OcrEngineManager.CreateEngine(OcrEngineType.LEAD, false))
29             {
30                 //Startup the LEADTOOLS OCR Engine
31                 ocrEngine.Startup(null, null, null, null);
32                 //Run the AutoRecognizeManager and specify PDF format
33                 ocrEngine.AutoRecognizeManager.Run(inputFile, outputFile, DocumentFormat.Pdf, null, null);
34                 Console.WriteLine($"OCR output saved to {outputFile}");
35             }
36         }
37     }
38 }
```

Above is an entire OCR application using LEADTOOLS

That's it. Clean and simple code that produces a fast and accurate OCR app backed by the most powerful OCR SDK. Download our free evaluation and see how easy it is to use LEADTOOLS for your document, medical, and multimedia projects.



Get Started Today

DOWNLOAD OUR FREE EVALUATION

[LEADTOOLS.COM](https://www.leadtools.com)



Mastering Blockchain

You may have noticed that *MSDN Magazine* has been shining a light on the topic of blockchain technology, which can leverage shared business processes and data across multiple, semi-trusted organizations. While cryptocurrencies were the first application for blockchain, the technology promises to have profound impacts on everything from financial services to inventory and supply chain management.

In March, Jonathan Waldman published his widely read feature, “Blockchain Fundamentals” (msdn.com/magazine/mt845650), which provides a great introduction to the workings of blockchain technology and its application beyond the arena of cryptocurrencies. Then last month Stefano Tempesta published a walk-through of Azure Blockchain Workbench (msdn.com/magazine/mt846726), which Microsoft debuted at the Build Conference in Seattle in May. The tool streamlines development of complex blockchain applications so organizations can focus on things that add value, like robust business logic and smart contracts, rather than scaffolding.

Next month, Waldman returns with a follow-up to his March feature, diving into topics like the transaction hash chain and proof-of-work and proof-of-stake consensus algorithms.

Now in this issue, Tempesta returns with his feature, “Decentralized Applications with Azure Blockchain as a Service,” which explores Microsoft’s effort to provide, as Tempesta puts it, “a rapid, low-cost, low-risk platform for building and deploying blockchain

applications.” Just as important, Blockchain as a Service takes the next step in enabling blockchains to interact with external data assets, so they can be applied to a wide variety of scenarios.

“Overall, Blockchain as a Service in Azure provides a level of integration among multiple Azure services that you cannot find in any other cloud offering,” Tempesta explains. “Azure Active Directory (AD), Key Value, Service Bus, App Service and SQL Server are all part of an ecosystem of services connected among each other in a secure way to guarantee the integrity of the blockchain.”

Tempesta says Microsoft has successfully leveraged the reliable and secure cloud infrastructure of Azure to position itself as a leader in the enterprise blockchain space.

Our work in this area is hardly done. Next month, Waldman returns with a follow-up to his March feature, diving into topics like the transaction hash chain and proof-of-work and proof-of-stake consensus algorithms. He’ll also explore what he calls the “inevitable formation” of blockchain forks and how they can be resolved. And Tempesta is eyeing a follow-up of his own that will focus on encryption and security.

Why all this coverage? As Waldman points out, because it’s needed.

“Based on feedback I received from my March article, it’s clear many readers want to learn about blockchain technologies, but have been disappointed by the prevalence of educational material that targets or advocates for a particular implementation,” he says. “By exploring what I call ‘core technical underpinnings,’ readers can build a mental map of what a blockchain looks like. Then they can overlay that image as they study specific blockchains or as they begin to design their own blockchain-powered applications.”

As blockchain solutions and technology continue to advance and evolve, our coverage of it will also. Is there an aspect of blockchain development and implementation you would like to see addressed in *MSDN Magazine*? E-mail me at mmeditor@microsoft.com and let me know.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2018 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.

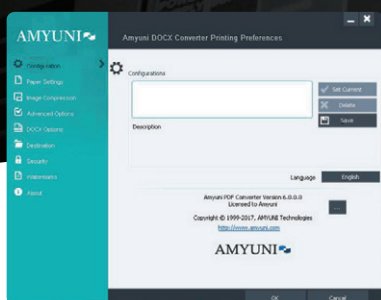


DOCXCONVERTER
For Windows

Free Demo at DOCXConverter.com

Amyuni DOCX Converter for Windows

Convert any document, including PDF documents, into DOCX format.
Enable editing of documents using Microsoft Word or other Office products.



A standalone desktop version, a server product for automated processing or an SDK for integration into third party applications.

Create

Create naturally editable DOCX documents with paragraph formatting and reflow of text

Convert

Convert images and graphics of multiple formats into DOCX shapes

OCR

Use OCR technology to convert non-editable text into real text

Extract

Extract headers and footers from source document and save them as DOCX headers and footers

Open

Open PDF documents with the integrated PDF viewer and quickly resave them to DOCX format

Configure

Configure the way the fonts are embedded into the DOCX file for optimal formatting

A virtual printer driver available for Windows 7 to Windows 10 and Windows Server 2008 to 2016

Powered by Amyuni Technologies:

Developers of the Amyuni PDF Converter and Amyuni PDF Creator products integrated into hundreds of applications and installed on millions of desktops and servers worldwide.

www.docxconverter.com

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.



EF Core 2.1 Query Types

EF Core 2.1 is here! And there are many great new features and improvements. Rather than taking over the entire magazine to tell you about them all, I'll share with you the new Query Type feature, which lets you more easily query the database without needing true entities with key properties to consume the results.

Prior to query types, it was possible to write queries against database views and to execute stored procedures with EF Core, but there were limitations. For the views, you had to rely on the fact that EF Core doesn't know the difference between a view and a table in your database. You could create entities that were part of your DbContext model, create DbSet for those entities and then write queries against those DbSet. But a lot of caveats came with that workflow, such as having to take care not to edit the resulting objects and accidentally causing SaveChanges to attempt to execute an update command, which would fail in the database unless your view was updatable. When executing stored procedures using the FromSql method, you were again required to tie the results to a true entity that was part of your data model, which meant adding extra types to your data model that really didn't need to be there.

The new Query Type enables easier paths to working with views, stored procedures and other means of querying your database. This is because the query type allows you to let EF Core interact with types that don't have key properties and map to database objects that don't have primary keys. EF has always been reliant on keys, so this is a big step for EF Core. Additionally, the query type will help you avoid any interaction with the change tracker, so you don't have to add in code to protect your application from inadvertent runtime exceptions related to entities that aren't updatable. You can even use query types to map to tables, forcing them to be read-only.

In this article I'm going to explore three capabilities enabled by query types:

- Querying against database views
- Another new feature called “defining queries”
- Capturing the results of FromSql queries with non-entity types

Vital to query types is letting the DbContext ModelBuilder know that a type should be recognized as a query type. You do that by creating a DbQuery property either in the context or with the modelBuilder.Query method. Both are new.

If you've used EF or EF Core at all, you should be familiar with DbSet, the EF class that allows you to query and update entities

of a particular type through a DbContext. DbQuery is a cousin to DbSet, wrapping non-entity types and allowing you to execute read-only queries against views and tables. And these types wrapped in a DbQuery are query types.

The EF Core convention for a DbQuery is similar to a DbSet in that EF Core expects the name of the DbQuery property to match the name of the database object to which it maps.

The new Query Type enables easier paths to working with views, stored procedures and other means of querying your database.

Two points you should be aware of are that migrations can't build views for you based on mappings, and EF Core can't reverse-engineer views (yet).

Mapping to and Querying a Database View

I'll use DbQuery for the first example—mapping to a database view and querying from it. This DbQuery presumes there's a class

Figure 1 The Entity Classes for the Sample Model

```
public class Magazine
{
    public int MagazineId { get; set; }
    public string Name { get; set; }
    public string Publisher { get; set; }
    public List<Article> Articles { get; set; }
}

public class Article
{
    public int ArticleId { get; set; }
    public string Title { get; set; }
    public int MagazineId { get; set; }
    public DateTime PublishDate { get; set; }
    public Author Author { get; set; }
    public int AuthorId { get; set; }
}

public class Author
{
    public int AuthorId { get; set; }
    public string Name { get; set; }
    public List<Article> Articles { get; set; }
}
```

Code download available at msdn.com/magazine/0718magcode.

already defined, as well as a view named `AuthorArticleCounts` in the database:

```
public DbQuery<AuthorArticleCount>
    AuthorArticleCounts { get; set; }
```

This alone will allow you to query a database view. Let's back up, though, to look at the model shown in **Figure 1**.

I'm using a simple model with three entities to manage publications: `Magazine`, `Article` and `Author`.

In my database, in addition to the `Magazines`, `Articles` and `Authors` tables, I have a view called `AuthorArticleCounts`, defined to return the name and number of articles an author has written:

```
SELECT
    a.AuthorName,
    Count(r.ArticleId) as ArticleCount
from Authors a
JOIN Articles r on r.AuthorId = a.AuthorId
GROUP BY a.AuthorName
```

I've also created the `AuthorArticleCount` class that matches the schema of the view results. In the class, I made the property setters private to make it clear that this class is read-only, even though EF Core won't ever attempt to track or persist data from a query type.

```
public class AuthorArticleCount
{
    public string AuthorName { get; private set; }
    public int ArticleCount { get; private set; }
}
```

With the database view in place and a class designed to consume its results, all I need to map them together is a `DbQuery` property in my `DbContext`—the same example I showed earlier:

```
public DbQuery<AuthorArticleCount> AuthorArticleCounts { get; set; }
```

Now EF Core will be happy to work with the `AuthorArticleCount` class, even though it has no key property, because EF Core understands this to be a query type. You can use it to write and execute queries against the database view.

For example, this simple LINQ query:

```
var results = _context.AuthorArticleCounts.ToList();
```

will cause the following SQL to be sent to my SQLite database:

```
SELECT "v"."ArticleCount", "v"."AuthorName"
FROM "AuthorArticleCounts" AS "v"
```

The results are a set of `AuthorArticleCount` objects, as shown in **Figure 2**.

And the `ChangeTracker` of the context used to execute the query is totally unaware of these objects.

This is a much nicer experience than past EF Core and Entity Framework implementations where database views were treated like tables, their results had to be entities and you had to take care not to accidentally track them with the change tracker.

It's possible to execute queries without predefining a `DbQuery` in the `DbContext` class. `DbSet` allows this, as well, with the `Set` method of a `DbContext` instance. For a `DbQuery`, you can write a query as:

```
var results = _context.Query<AuthorArticleCount>().ToList();
```

```
results [List]: Count = 2
[0]: {AuthorArticleCount}
    ArticleCount [int]: 1
    AuthorName [string]: "Andrew Marantz"
[1]: {AuthorArticleCount}
    ArticleCount [int]: 2
    AuthorName [string]: "Julie Lerman"
```

Figure 2 Results of One-to-One Query

```
results [List]: Count = 3
[0]: {Publications.AuthorArticleCount}
[1]: {Publications.AuthorArticleCount}
    ArticleCount [int]: 2
    Author: {Publications.Author}
        Articles [List]: null
        AuthorId [int]: 1
        Name [string]: "Julie Lerman"
        AuthorName [string]: "Julie Lerman"
[2]: {Publications.AuthorArticleCount}
```

Figure 3 Results of Eager Loading a One-to-One Relationship Between a Query Type and an Entity

Configuring Query-Type Mappings

This `DbQuery` worked easily because everything follows convention. When `DbSets` and their entities don't follow EF Core conventions, you use the Fluent API or data annotations to specify the correct mappings in the `OnModelCreating` method. And you begin by identifying which entity in the model you want to affect using the `ModelBuilder`'s `Entity` method. Just as `DbSet` gained a cousin in `DbQuery`, the `Entity` method also has a new cousin: `Query`. Here's an example of using the `Query` method to point the `AuthorArticleCounts` `DbQuery` to a view of a different name, using the new `ToView` method (similar to the `ToTable` method):

```
modelBuilder.Query<AuthorArticleCount>().ToView(
    "View_AuthorArticleCounts");
```

The `Query<T>` method returns a `QueryTypeBuilder` object. `ToView` is an extension method. There are a number

of methods you can use when refining the query type. `QueryTypeBuilder` has a subset of `EntityTypeBuilder` methods: `HasAnnotation`, `HasBaseType`, `HasOne`, `HasQueryFilter`, `IgnoreProperty` and `UsePropertyAccessMode`. There's a nice explanation about `ToView` and `ToTable` highlighted as a Tip in the Query Types documentation that I recommend (bit.ly/2kmQhV8).

Query Types in Relationships

Notice the `HasOne` method. It's possible for a query type to be a dependent (aka "child") in a one-to-one or one-to-many relationship with an entity, although not with another query type. Also note that query types aren't nearly as flexible as entities in relationships, which is reasonable in my opinion. And you have to set up the relationships in a particular way.

It's possible to execute queries without predefining a `DbQuery` in the `DbContext` class.

I'll start with a one-to-one relationship between the `Author` entity and `AuthorArticleCount`. The most important rules for implementing this are:

- The query type must have a navigation property back to the other end of the relationship.
- The entity can't have a navigation property to the query type.

In the latter case, if you were to add an `AuthorArticleCount` property to `Author`, the context would think the `AuthorArticleCount` is an entity and the model builder would fail.

I've enhanced the model with two changes:

First, I modified the `AuthorArticleCount` to include an `Author` property:

```
public Author Author { get; private set; }
```

Then I added a one-to-one mapping between `Author` and `AuthorArticleCount`:

```
modelBuilder.Query<AuthorArticleCount>().  
    .HasOne<Author>().  
    .WithOne();
```

Now I can execute LINQ queries to eager load the `Author` navigation property, for example:

```
var results = _context.AuthorArticleCounts.Include("Author").ToList();
```

The results are shown in **Figure 3**.

Query Types in a One-to-Many Relationship

A one-to-many relationship also requires that the query type be the dependent end, never the principal (aka parent). To explore this, I created a new view over the `Articles` table in the database called `ArticleView`:

```
CREATE VIEW ArticleView as select Title, PublishDate, MagazineId from Articles;
```

And I created an `ArticleView` class:

```
public class ArticleView  
{  
    public string Title { get; set; }  
    public Magazine Magazine { get; set; }  
    public int MagazineId { get; set; }  
    public DateTime PublishDate { get; set; }  
}
```

ToQuery allows you to define a query directly in the DbContext, and such a query is referred to as a “defining query.”

Finally, I specified that `ArticleView` is a query type and defined its relationship with the `Magazine` entity, where a `Magazine` can have many `ArticleViews`:

```
modelBuilder.Query<ArticleView>().HasOne(a => a.Magazine).WithMany();
```

Now I can execute a query that retrieves graphs of data. I’ll use an `Include` method again. Remember that there’s no reference to the query type in the `Magazine` class, so you can’t query for a graph of a magazine with its `ArticleViews` and see those graphs. You can only navigate from `ArticleView` to `Magazine`, so this is the type of query you can perform:

```
var articles = _context.Query<ArticleView>().  
    Include(m => m.Magazine).ToList();
```

Notice that I didn’t create a `DbQuery` so I’m using the `Query` method in my query.

The API documentation for `HasOne`, which you’ll find at bit.ly/2lm8UqR, provides more detail about using this method.

The New Defining Query Feature

Besides `ToView`, there’s one other new method on `QueryTypeBuilder` that never existed on `EntityTypeBuilder`, and that’s `ToQuery`.

`ToQuery` allows you to define a query directly in the `DbContext`, and such a query is referred to as a “defining query.” You can write LINQ queries and even use `FromSql` when composing defining queries. Andrew Peters from the EF team explains that, “One use of `ToQuery` is for testing with the in-memory provider. If my app is using a database view, I can also define a `ToQuery` that will be used only if I’m targeting in-memory. In this way I can simulate the database view for testing.”

To start, I created the `MagazineStatsView` class to consume the results of the query:

```
public class MagazineStatsView  
{  
    public MagazineStatsView(string name, int articleCount, int authorCount)  
    {  
        Name=name;  
        ArticleCount=articleCount;  
        AuthorCount=authorCount;  
    }  
    public string Name { get; private set; }  
    public int ArticleCount { get; private set; }  
    public int AuthorCount { get; private set; }  
}
```

I then created a defining query in `OnModelCreating` that queries the `Magazine` entities, and builds `MagazineStatsView` objects from the results:

```
modelBuilder.Query<MagazineStatsView>().ToQuery(  
    () => Magazines.Select( m => new MagazineStatsView(  
        m.Name,  
        m.Articles.Count,  
        m.Articles.Select(a => a.AuthorId).Distinct().Count()  
    )  
);
```

I could also create a `DbQuery` to make my new defining query a little more discoverable, but I wanted you to see that I can still use this without an explicit `DbQuery`. Here’s a LINQ query for `MagazineStatsView`. It will always be handled by the defining query:

```
var results = _context.Query<MagazineStatsView>().ToList();
```

Based on the data I’ve used to seed the database, the results of the query, shown in **Figure 4**, correctly show two articles and one unique author for *MSDN Magazine*, and two articles with two unique authors for *The New Yorker*.

Capture FromSql Results in Non-Entity Types

In previous versions of Entity Framework, it was possible to execute raw SQL and capture those results in random types. We are closer to being able to perform this type of query thanks to query types. With EF Core 2.1, the type you want to use to capture the results of raw SQL queries doesn’t have to be an entity, but it still has to be known by the model as a query type.

There’s one exception to this, which is that it’s possible (with a lot of limitations) to return anonymous types. Even this limited support can still be useful, so it’s worth being aware of. Here’s a query that returns an anonymous type using `FromSql` and a raw SQL query:

```
context.Authors.FromSql("select  
    authorid,authorname from authors").ToList();
```

Returning anonymous types by querying entities only works when the projection includes the primary key of the type represented by the `DbSet`. If I didn’t include `AuthorId`, a runtime error would complain about `AuthorId` not being in the projection.

```
results [List]: Count = 2  
└─ [0]: {MagazineStatsView}  
    ArticleCount [int]: 2  
    AuthorCount [int]: 1  
    Name [string]: "MSDN Magazine"  
└─ [1]: {MagazineStatsView}  
    ArticleCount [int]: 2  
    AuthorCount [int]: 2  
    Name [string]: "New Yorker"
```

Figure 4 Results of Querying with a Defining Query

Or if I began with context.Magazines.FromSql with the same query I just showed you, the runtime error would complain about MagazineId not being available.

A better use of this feature is to predefine a type and make sure the DbContext is aware of that type, either by defining a DbQuery or specifying modelBuilder.Query for the type in OnModelCreating. Then you can use FromSql to query and capture the results. As a somewhat contrived example, or perhaps I should say even more contrived than some of the examples I've used already, here's a new class, Publisher, that's not an entity or part of my PublicationsContext:

```
public class Publisher
{
    public string Name { get; private set; }
    public int YearIncorporated { get; private set; }
}
```

It, too, is a read-only class, as I have another application where I maintain Publisher data.

I created a DbQuery<Publisher> named Publishers in my context, and now I can use that to execute raw SQL query:

```
var publishers=context.Publishers
    .FromSql("select name, yearfounded from publishers")
    .ToList();
```

Raw SQL can also be a call to execute a stored procedure. As long as the schema of the results match the type (in this case, Publisher), you can do that, even passing in parameters.

If you've been holding off
on using EF Core until it was
production-ready, the time
has finally come.

Putting the Polish on EF Core

If you've been holding off on using EF Core until it was production-ready, the time has finally come. EF Core 2.0 made a great leap in features and functionality, and version 2.1 now includes features that put a real polish on the product. The wait for features from EF6 to appear in EF Core has been due in part to the fact that the EF team has not just copied the old implementations but found smarter, more functional implementations. Query types are a great example of this, compared to the way that views and raw SQL were supported in earlier versions of Entity Framework. Be sure to check out the other new features in EF Core 2.1 by reading the "New Features in EF Core 2.1" section of the EF Core documentation at bit.ly/2lhyHQR. ■

JULIE LERMAN is a Microsoft Regional Director, Microsoft MVP, software team coach and consultant who lives in the hills of Vermont. You can find her presenting on data access and other topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework," as well as a Code First and a DbContext edition, all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at julieme/PS-Videos.

THANKS to the following Microsoft technical expert for reviewing this article:
Andrew Peters

msdnmagazine.com



Automate and add interactivity to your PDF applications

.NET IMAGING and PDF

- ☒ Interactive form field
- ☒ JavaScript actions
- ☒ Barcode and Signature field
- ☒ Annotation and Content editing



Professional SDK for building document management apps

- Image Viewer for .NET, WPF and WEB
- 100+ Image Processing and Document Cleanup commands
- PDF Reader, Writer, Visual Editor
- Image Annotations
- JBIG2 and JPEG2000 codecs
- OCR and Document Recognition
- Forms Processing and OMR
- DICOM decoder
- Barcode Reader and Generator
- TWAIN scanning

Free evaluation version
Royalty free licensing

www.vintasoft.com

VintaSoft® is a registered trademark
of VintaSoft Ltd.



Visual Studio[®] LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

August 13 – 17, 2018
Redmond, WA

Microsoft Headquarters



Yesterday's Knowledge; Tomorrow's Code!

Visual Studio Live! (VSLive!™) is celebrating 25 years of coding innovation in 2018! From August 13 – 17, developers, software architects, engineers, designers and more will come together at Microsoft Headquarters for 5 days of unbiased education on the Microsoft Platform. Hone your skills in **Visual Studio**, **ASP.NET Core**, **AngularJS**, **SQL Server**, and so much more. Plus, you can eat lunch with the Blue Badges, rub elbows with Microsoft insiders, explore the campus, all while expanding your ability to create better apps!

DEVELOPMENT TOPICS INCLUDE:



Visual Studio / .NET



Native Client



Web Client



Cloud Computing



Software Practices



ASP.NET / Web Server



ALM / DevOps



Database & Analytics



Microsoft Sessions



vslive.com/redmond

Save \$300
Early Bird Savings Ends 7/13

Use promo code MSDN

EVENT SPONSOR



PLATINUM SPONSOR



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server
START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, August 13, 2018 <i>(Separate entry fee required)</i>					
7:00 AM	8:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries					
8:00 AM	12:00 PM	M01 Workshop: Build a Modern ASP.NET App in the Cloud with a full CI/CD Pipeline in VSTS - Brian Randell		M02 Workshop: Developer Dive into SQL Server 2016 - Leonard Lobel		M03 Workshop: Distributed Cross-Platform Application Architecture - Rockford Lhotka and Jason Bock	
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center					
2:00 PM	5:30 PM	M01 Workshop Continues		M02 Workshop Continues		M03 Workshop Continues	
7:00 PM	9:00 PM	Dine-A-Round Dinner					
START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, August 14, 2018					
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	T01 Angular 101 - Deborah Kurata	T02 Xamarin: The Future of App Development - James Montemagno	T03 Cloud Oriented Programming - Vishwas Lele	T04 Busting .NET Myths - Jason Bock	T05 A DevOps Journey - Abel Wang	
9:30 AM	10:45 AM	T06 Write Object-Oriented JavaScript with TypeScript - Rachel Appel	T07 Netstandard: Reuse C# Code Across Windows, Mac, Linux, iOS, Android - Rockford Lhotka	T08 Microservices with ACS (Managed Kubernetes) - Vishwas Lele	T09 Get Started with Git - Robert Green	T10 DevOps for the SQL Server Database - Brian Randell	
10:45 AM	11:15 AM	Sponsored Break - Visit Exhibitors - Foyer					
11:15 AM	12:15 PM	KEYNOTE: To Be Announced					
12:15 PM	1:30 PM	Lunch - McKinley / Visit Exhibitors - Foyer					
1:30 PM	2:45 PM	T11 Angular Component Communication - Deborah Kurata	T12 Tips & Tricks for Xamarin Development - James Montemagno	T13 Modern SQL Server Security Features for Developers - Leonard Lobel		T14 To Be Announced	T15 Microsoft Session To Be Announced
3:00 PM	4:15 PM	T16 Build Data Driven Web Apps Using ASP.NET Core - Rachel Appel	T17 To Be Announced	T18 Introduction to Azure Cosmos DB - Leonard Lobel		T19 Building A Development Culture of Collaboration - Justin Collier	T20 Microsoft Session To Be Announced
4:15 PM	5:45 PM	Microsoft Ask the Experts & Exhibitor Reception – Attend Exhibitor Demos					
START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, August 15, 2018					
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	W01 Assembling the Web - A Tour of WebAssembly - Jason Bock	W02 Mobile App Development for the Web Developer - Ben Hoelting	W03 - SQL Server 2017 - Intelligence Built-in - Scott Klein	W04 Azure DevOps with VSTS, Docker, and K8 - Brian Randell	W05 Microsoft Session To Be Announced	
9:30 AM	10:45 AM	W06 Getting Pushy with SignalR and Reactive Extensions - Jim Wooley	W07 Cross-Platform App Development Using Xamarin and CSLA .NET - Rockford Lhotka	W08 An Architect's Guide to Data Science - Becky Isserman		W09 Use Visual Studio to Scale Agile in Your Enterprise - Richard Hundhausen	W10 Microsoft Session To Be Announced
11:00 AM	12:00 PM	GENERAL SESSION: To Be Announced					
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors					
1:30 PM	2:45 PM	W11 Building Reactive Client Experiences with RxJs - Jim Wooley	W12 Radically Advanced XAML: Dashboards, Timelines, Animation, and More - Billy Hollis	W13 Knockout: R vs Python for Data Science - Becky Isserman		W14 Develop on Cadence, Release on Demand - Richard Hundhausen	W15 Microsoft Session To Be Announced
2:45 PM	3:15 PM	Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)					
3:15 PM	4:30 PM	W16 When It Isn't as Simple as js -> .ts - Garvice Eakins	W17 Programming with the Model-View-ViewModel Pattern - Miguel Castro	W18 Busy Developer's Guide to NoSQL - Ted Neward		W19 Visualizing the Backlog with User Story Mapping - Philip Japikse	W20 Data Science for Developers - Aashish Bhateja
6:15 PM	9:00 PM	Set Sail! VSLive's Seattle Sunset Cruise - Advanced Reservation & \$20 Fee Required					
START TIME	END TIME	Visual Studio Live! Day 3: Thursday, August 16, 2018					
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	TH01 The Whirlwind Tour of Authentication and Authorization with ASP.NET Core - Chris Klug	TH02 Getting Started Debugging C# Application - Paul Sheriff	TH03 Leaders Are Made, Not Born - Philip Japikse		TH04 Using The Microsoft Cognitive Custom Vision Service - Michael Washington	TH05 Lessons Learned from Making Resilient Apps with Azure Mobile App Services - Matthew Soucoup
9:30 AM	10:45 AM	TH06 Introduction to Web Pack - Chris Klug	TH07 C# 7.x Like a Boss! - Adam Tuliper	TH08 The Role of an Architect - Ted Neward		TH09 Google Home Meets .NET Containers on Google Cloud - Mete Atamel	TH10 Microsoft Session To Be Announced
11:00 AM	12:15 PM	TH11 JavaScript Patterns for the C# Developer - Ben Hoelting	TH12 I'll Get Back to You: Task, Await, and Asynchronous Methods - Jeremy Clark	TH13 Demystifying Microservice Architecture - Miguel Castro		TH14 Building Business Applications Using Bots - Michael Washington	TH15 Microsoft Session To Be Announced
12:15 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center					
2:00 PM	3:15 PM	TH16 Utilizing the MVVM Design Pattern in WPF - Paul Sheriff	TH17 Getting to the Core of the .NET Standard - Adam Tuliper	TH18 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - Jeremy Clark		TH19 Wash, Rinse, Repeat: Writing Skills for Both Alexa and Cortana - Christine Flora	TH20 Microsoft Session To Be Announced
3:30 PM	4:45 PM	TH21 WPF Styles, Resources and Templates, Oh My! - Paul Sheriff	TH22 Building Apps with Microsoft Graph and Visual Studio - Robert Green	TH23 How to Interview a Developer - Billy Hollis		TH24 The Complete Package: Creating a Deployable Solution for Microsoft Teams - Christine Flora	TH25 Microsoft Session To Be Announced
START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, August 17, 2018 <i>(Separate entry fee required)</i>					
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries					
8:00 AM	5:00 PM	F01 Workshop: UX Design for Developers - Billy Hollis			F02 Workshop: Web Development in 2018 - Chris Klug		

Speakers and sessions subject to change

CONNECT WITH US



twitter.com/vslive - @VSLive



facebook.com - Search "VSLive"



linkedin.com - Join the "Visual Studio Live" group!

vslive.com/redmond



How To Be MEAN: Dynamically Angular

Welcome back again, MEANers.

In my last column, “How To Be MEAN: Reactive Programming” (msdn.com/magazine/mt846724), I examined the reactive forms module of Angular, which offers a different way to construct a form and respond to the events the user creates within that. At the end of that article, I posed a question: What if I have a situation where there’s a fairly large number of controls, or the controls created need to change based on the changing nature of the model object underneath? This is one of those cases where, frankly, relying on the traditional “template” system that Angular employs won’t cut it. If I have to create a template for every possible combination of possibilities, it’s going to be a really long day.

Let’s assume, for the moment, that a conference wishes to build a poll system for attendees to evaluate speakers, talks, the conference venue, the Web site, you name it. They want to be able to roll out new questions pretty quickly, potentially even during the conference itself should the need arise. This means, then, that I want Web pages that know how to generate fields based on the question types being asked, and those question types will be coming from an external source (such as a JSON service or even a file).

It’s not magic. In any GUI system (Web-based or otherwise) that supports the construction of controls via a runtime construct (such as being able to “new” up the controls themselves), this is a reasonable and quite doable thing. It’s certainly doable in Angular: I can build a system that builds a form entirely off of model objects and the associated (implicit or explicit) metadata therein.

So, to continue with the questionnaire example, if I build a simple Angular service that knows how to obtain a series of “question” objects, an associated Angular form can take some or all of those objects and construct the corresponding collection of form elements to present the questions and capture the answers, presumably for storage somewhere. The key to all of this will be the FormGroup and FormControl that Angular uses to represent those controls at runtime.

Dynamic Model

Let’s start with a base class for all questions, which will help capture some common behavior I expect (and will need) for any question and its related control. Here’s the code for that:

```
export type ControlType = "textbox" | "dropdown";

export abstract class Question {
  constructor(public value: string = '',
    public key: string = '',
    public label: string = '',
    public required: boolean = false,
    public controlType: ControlType = 'textbox')
  { }
}
```

Most of this is going to be pretty straightforward, because most of the class here is just properties (what the patterns folks sometimes call a DTO, or Data Transfer Object), but the key element is going to be the controlType field. It will be the descriptor that corresponds to what HTML constructs I generate. Currently, it has all of two possibilities: a textbox (allowing for open-ended text entry) or a dropdown (a single-item selected from a bounded range of possibilities).

Equally obvious, Question is an abstract class, because I expect derived types to be created here, one for each type of Question. The code for TextboxQuestion looks like this:

```
export class TextboxQuestion extends Question {
  constructor(value: string = '',
    key: string = '',
    label: string = '',
    required: boolean = false,
    public type: string = '') {
    super(value, key, label, required, 'textbox');
  }
}
```

And the code for DropdownQuestion like this:

```
export class DropdownQuestion extends Question {
  constructor(value: string = '',
    key: string = '',
    label: string = '',
    required: boolean = false,
    public options: {key: string, value: string}[] = []) {
    super(value, key, label, required, 'dropdown');
  }
}
```

Each question passes the set of base parameters up to its parent, and each adds one thing to the mix. In the case of TextboxQuestion, it adds a type parameter for the textbox in case I want to signify that this is a password or e-mail textbox. In the case of DropdownQuestion, it adds an array of key/value pairs to use as the dropdown possibilities.

Next, however, I have to figure out how to turn these into FormControl and FormGroup objects. Arguably, according to the way

Figure 1 The AppComponent

```
@Component({
  selector: 'app-root',
  template: `
    <div>
      <h2>How was our conference?</h2>
      <app-questionnaire [questions]="questions"></app-questionnaire>
    </div>
  `,
  providers: [QuestionService]
})
export class AppComponent {
  questions: Question[];

  constructor(service: QuestionService) {
    this.questions = service.getQuestions();
  }
}
```

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

September 17 – 20, 2018
Renaissance Chicago

Chicago

Look Back to
Code Forward.

#VSLIVE25

HELP US CELEBRATE 25 YEARS
OF CODING INNOVATION!



Visual Studio Live! (VSLive!™) is thrilled to be returning to Chicago this September where developers, software architects, engineers and designers will tackle training on the hottest topics (like .NET Core, Angular, VS2017), debate with industry and Microsoft insiders and network with your peers. Come experience the education, knowledge-share and networking at #VSLive25.

DEVELOPMENT TOPICS INCLUDE:



DevOps in the
Spotlight



Cloud, Containers
and Microservices



AI, Data and
Machine Learning



Developing New
Experiences



Delivery and Deployment



.NET Core and More



Full Stack Web Deployment

CONNECT WITH US



twitter.com/
vslive – @VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

Register Now to
Save Up to \$400!
Use promo code MSDN



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



vslive.com/chicago

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

October 7 – 11, 2018
Hilton San Diego Resort

San Diego

**Code Again for
the First Time!**

#VSLIVE25

DEVELOPMENT TOPICS INCLUDE:



DevOps in the
Spotlight



Cloud, Containers
and Microservices



AI, Data and
Machine Learning



Developing New
Experiences



Delivery and Deployment



.NET Core and More



Full Stack Web Deployment



Hands-On Labs

CODE WITH US IN SUNNY SAN DIEGO!



For the **FIRST TIME EVER** in our 25 year history, Visual Studio Live! is heading to San Diego, CA for up to 5 days of practical, unbiased, Developer training, including **NEW** intense hands-on labs. Join us as we dig into the latest features of Visual Studio 2017, including ASP.NET Core, Angular, Xamarin, UWP and more. Help us celebrate 25 years of coding innovation and experience the education, networking and knowledge-share at #VSLive25.

CONNECT WITH US



twitter.com/
vslive – @VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

**Register Now to
Save \$300!**
Use promo code MSDN



SUPPORTED BY



PRODUCED BY



vslive.com/sandiego

Angular thinks about design, that could be a standalone service, but it makes more sense to me to make it a part of the Question class, as a static method. (If I ever add a new Question type, it's this method that needs to be updated, so it makes more sense to me to keep them all grouped within the same module.) From the code side, creating the requisite FormControl objects is pretty straightforward, as follows:

```
export abstract class Question {

  public static toFormGroup(questions: Question[]): FormGroup {
    let group: any = {};

    questions.forEach(question => {
      group[question.key] =
        question.required ? new FormControl(question.value, Validators.required)
          : new FormControl(question.value);
    });
    return new FormGroup(group);
  }
  // ...
}
```

This method basically takes an array of Questions and turns them into an array of FormControl objects nestled inside of a FormGroup object. From this side of things, notice that the only real question is whether the control is required; any other display logic will need to be captured inside the template.

Dynamic Display

I also need to start thinking about the Angular UI components involved here; fundamentally, a poll or questionnaire is made up of one or more questions, so I'll use that as the working model: a QuestionnaireComponent uses some number of QuestionComponents, and each QuestionComponent will have as input a Question object.

It feels a little simpler to start from the top and work my way down, so let's do that. First off, I have the AppComponent that will display the questionnaire, in this case on its own, as shown in **Figure 1**.

This code offers up the perfect component scenario. I just use it, and have a service that knows how to provide the input the component needs, so the code stays light, simple and easily intuitive to any Angular developer.

Next, let's look at the QuestionnaireComponent, as shown in **Figure 2**.

Again, the approach is pretty straightforward and simple. The QuestionnaireComponent takes an array of Questions as its input, and uses the FormGroup to match up to the form to be built in the template. **Figure 3** shows this.

Generally speaking, the payload would be uploaded via HTTP through an Angular service, presumably to be stored in a database, but that's taking the example a little out of scope. Here, displaying serves to demonstrate that the data is validated, captured and prepped for distribution.

Of course, I still have to build the individual question elements within the form, and that falls to the QuestionComponent code, shown right here:

```
@Component({
  selector: 'app-question',
  templateUrl: './question.component.html'
})
export class QuestionComponent {
  @Input() question: Question;
  @Input() form: FormGroup;
  get isValid() { return this.form.controls[this.question.key].valid; }
}
```

Notice that the QuestionComponent takes as input the FormGroup to which it (logically) belongs; I could try to find a different means by which to obtain the FormControl (for the isValid property implementation), but this way works and helps keep things simple.

The template for this component is where the real magic of the dynamic form creation takes place. Thanks to a judicious ngSwitch on the Question object's controlType, I can build the HTML element pretty simply, as shown in **Figure 4**.

As you can see, it's pretty elegant as these things go. I switch on the controlType property, and depending on whether this is a dropdown or a textbox type question, build different HTML.

Last, I just need a QuestionService that serves up some questions, which, again, would usually do so from some external

Figure 2 The QuestionnaireComponent

```
@Component({
  selector: 'app-questionnaire',
  templateUrl: './questionnaire.component.html'
})
export class QuestionnaireComponent implements OnInit {

  @Input() questions: Question[] = [];
  form: FormGroup;
  payload = '';

  ngOnInit() {
    this.form = Question.toFormGroup(this.questions);
  }

  onSubmit() {
    this.payload = JSON.stringify(this.form.value);
  }
}
```

Figure 3 Preparing to Build the Form with FormGroup

```
<div>
  <form (ngSubmit)="onSubmit()" [formGroup]="form">
    <div *ngFor="let question of questions" class="form-row">
      <app-question [question]="question" [form]="form"></app-question>
    </div>

    <div class="form-row">
      <button type="submit" [disabled]="!form.valid">Save</button>
    </div>
  </form>

  <div *ngIf="payload" class="form-row">
    <strong>Saved the following values</strong><br>{{payload}}
  </div>
</div>
```

Figure 4 Building the HTML Element

```
<div [formGroup]="form">
  <label [attr.for]="question.key">{{question.label}}</label>
  <div [ngSwitch]="question.controlType">

    <input *ngSwitchCase="'textbox'" [formControlName]="question.key"
      [id]="question.key" [type]="question.type">

    <select *ngSwitchCase="'dropdown'" [formControlName]="question.key"
      [id]="question.key">
      <option *ngFor="let opt of question.options" [value]="opt.key">
        {{opt.value}}
      </option>
    </select>
  </div>

  <div class="errorMessage" *ngIf="!isValid">{{question.label}} is required</div>
</div>
```


Figure 5 Getting Questions from QuestionService

```
@Injectable()
export class QuestionService {

  getQuestions() {
    return [
      new TextBoxQuestion('', 'firstName',
        'Speaker\'s First name', true),

      new DropdownQuestion('', 'enjoyment',
        'How much did you enjoy this speaker\'s talk?',
        false,
        [
          {key: 'great', value: 'Great'},
          {key: 'good', value: 'Good'},
          {key: 'solid', value: 'Solid'},
          {key: 'uninspiring', value: 'Uninspiring'},
          {key: 'wwyt', value: 'What Were You Thinking?'}
        ]
      ),
    ];
  }
}
```

resource like a file or a server-side API. In this particular example, the service pulls the question from memory, as depicted in Figure 5.

Obviously, in a real questionnaire, a few more questions are likely, but this example gets the point across.

Wrapping Up

The real question pertaining to any sort of system like this is its extensibility: Can I add new questionnaires without requiring significant modification? Obviously, the QuestionnaireService is the key there—

so long as it can yield back different arrays of Question objects, I have an infinite number of questionnaires I can ask our conference attendees. The only restriction is the kinds of questions I can ask right now, being limited to either multiple-choice or open-ended-text answers.

That raises a second question: How hard would it be to add new types of questions into the system, such as a ratings control with discrete numeric values? To do so would require the creation of a new Question subclass (RatingsQuestion) with the numeric range to use, a new ControlType enumeration value for the template to switch on, and modifying the QuestionComponent template to switch on the new enumeration value and display the HTML accordingly (however that would look). Everything else would remain untouched, which is the goal of any component technology—keep the client unaware of any structural changes unless they choose to take advantage of the new features.

Angular readers will be itching to give this whole concept a spin, so I'll bring things to a close here. However, there's one more necessary bit we need to go over before we can wrap up our Angular coverage, so we'll hit that next time. Until then, happy coding! ■

TED NEWARD is a Seattle-based polytechnology consultant, speaker, and mentor, currently working as the director of Engineering and Developer Relations at Smartsheet.com. He has written a ton of articles, authored and co-authored a dozen books, and speaks all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.

THANKS to the following technical expert: Garvice Eakins (Smartsheet)



SUPER-FAST AND ADVANCED CHARTS

LightningChart®

- WPF and WinForms
- Real-time scrolling up to 2 billion points in 2D
- Hundreds of examples
- On-line and off-line maps
- Advanced Polar and Smith charts
- Outstanding customer support



2D charts - 3D charts - Maps - Volume rendering - Gauges

www.LightningChart.com/ms

**TRY FOR
FREE**



Machine Learning with IoT Devices on the Edge

James McCaffrey

Imagine that, in the not too distant future, you're the designer of a smart traffic intersection. Your smart intersection has four video cameras connected to an Internet of things (IoT) device with a small CPU, similar to a Raspberry Pi. The cameras send video frames to the IoT device, where they're analyzed using a machine learning (ML) image-recognition model and control instructions are then sent to the traffic signals. One of the small IoT devices is connected to Azure Cloud Services, where information is logged and analyzed offline.

This is an example of ML on an IoT device on the edge. I use the term *edge device* to mean anything connected to the cloud, where cloud refers to something like Microsoft Azure or a company's remote servers. In this article, I'll explain two ways you can design ML on the edge. Specifically, I'll describe how to write a custom model and IO function for a device, and how to use the Microsoft Embedded Learning Library (ELL) set of tools to deploy an optimized ML model to a device on the edge. The custom IO approach is currently, as I write this article, the most common way to deploy an ML model to an IoT device. The ELL approach is forward-looking.

The ELL system is an open source project and is in the very early stages of development. Therefore, some of the information about ELL in this article may have changed by the time you read this article.

This article discusses:

- What a machine learning (ML) model consist of
- Deploying a standard ML model to an IoT device
- Writing a custom code solution
- Using the Microsoft Embedded Learning Library

Technologies discussed:

Visual Studio Code, Microsoft Embedded Learning Library (ELL), Python, Microsoft CNTK library

Code download available at:

msdn.com/magazine/0718magcode

Even if you're not working with ML on IoT devices, there are at least three reasons why you might want to read this article. First, the design principles involved generalize to other software development scenarios. Second, it's quite possible that you'll be working with ML and IoT devices relatively soon. Third, you may just find the techniques described here interesting in their own right.

Why does ML need to be on the IoT edge? Why not just do all processing in the cloud? IoT devices on the edge can be very inexpensive, but they often have limited memory, limited processing capability and a limited power supply. In many scenarios, trying to perform ML processing in the cloud has several drawbacks.

Latency is often a big problem. In the smart traffic intersection example, a delay of more than a fraction of a second could have disastrous consequences. Additional problems with trying to perform ML in the cloud include reliability (a dropped network connection is typically impossible to predict and difficult to deal with), network availability (for example, a ship at sea may have connectivity only when a satellite is overhead) and privacy/security (when, for example, you're monitoring a patient in a hospital.)

This article doesn't assume you have any particular background or skill set but does assume you have some general software development experience. The demo programs described in this article (a Python program that uses the CNTK library to create an ML model, a C program that simulates IoT code and a Python program that uses an ELL model) are too long to present here, but they're available in the accompanying file download.

What Is a Machine Learning Model?

In order to understand the issues with deploying an ML model to an IoT device on the edge, you must understand exactly what an ML model is. Very loosely speaking, an ML model is all the information needed to accept input data, make a prediction and generate output data. Rather than try to explain in the abstract, I'll illustrate the ideas using a concrete example.

Take a look at the screenshot in **Figure 1** and the diagram in **Figure 2**. The two figures show a neural network with four input nodes, five hidden layer processing nodes and three output layer nodes. The

input values are (6.1, 3.1, 5.1, 1.1) and the output values are (0.0321, 0.6458, 0.3221). **Figure 1** shows how the model was developed and trained. I used Visual Studio Code, but there are many alternatives.

This particular example involves predicting the species of an iris flower using input values that represent sepal (a leaf-like structure) length and width and petal length and width. There are three possible species of flower: setosa, versicolor, virginica. The output values can be interpreted as probabilities (note that they sum to 1.0) so, because the second value, 0.6458, is largest, the model's prediction is the second species, versicolor.

In **Figure 2**, each line connecting a pair of nodes represents a weight. A weight is just a numeric constant. If nodes are zero-base indexed, from top to bottom, the weight from input[0] to hidden[0] is 0.2680 and the weight from hidden[4] to output[0] is 0.9381.

Each hidden and output node has a small arrow pointing into the node. These are called biases. The bias for hidden[0] is 0.1164 and the bias for output[0] is -0.0466.

You can think of a neural network as a complicated math function because it just accepts numeric input and produces numeric output. An ML model on an IoT device needs to know how to compute output. For the neural network in **Figure 2**, the first step is to compute the values of the hidden nodes. The value of each hidden node is the hyperbolic tangent (tanh) function applied to the sum of the products of inputs and associated weights, plus the bias. For hidden[0] the calculation is:

```
hidden[0] = tanh((6.1 * 0.2680) + (3.1 * 0.3954) +
                (5.1 * -0.5503) + (1.1 * -0.3220) + 0.1164)
            = tanh(-0.1838)
            = -0.1817
```

Hidden nodes [1] through [4] are calculated similarly. The tanh function is called the hidden layer activation function. There are other activation functions that can be used, such as logistic sigmoid and rectified linear unit, which would give different hidden node values.

After the hidden node values have been computed, the next step is to compute preliminary output node values. A preliminary output node value is just the sum of products of hidden nodes and associated hidden-to-output weights, plus the bias. In other words, the same calculation as used for hidden nodes, but without the activation function. For the preliminary value of output[0] the calculation is:

```
o_pre[0] = (-0.1817 * 0.7552) + (-0.0824 * -0.7297) +
            (-0.1190 * -0.6733) + (-0.9287 * 0.9367) +
            (-0.9081 * 0.9381) + (-0.0466)
            = -1.7654
```

The values for output nodes [1] and [2] are calculated in the same way. After the preliminary values of the output nodes have been computed, the final output node values can be converted to probabilities using the softmax activation function. The softmax function is best explained by example. The calculations for the final output values are:

```
sum = exp(o_pre[0]) + exp(o_pre[1]) + exp(o_pre[2])
    = 0.1711 + 3.4391 + 1.7153
    = 5.3255
```

```
output[0] = exp(o_pre[0]) / sum
          = 0.1711 / 5.3255 = 0.0321
```

```
output[1] = exp(o_pre[1]) / sum
          = 3.4391 / 5.3255 = 0.6458
```

```
output[2] = exp(o_pre[2]) / sum
          = 1.7153 / 5.3255 = 0.3221
```

As with the hidden nodes, there are alternative output node activation functions, such as the identity function.

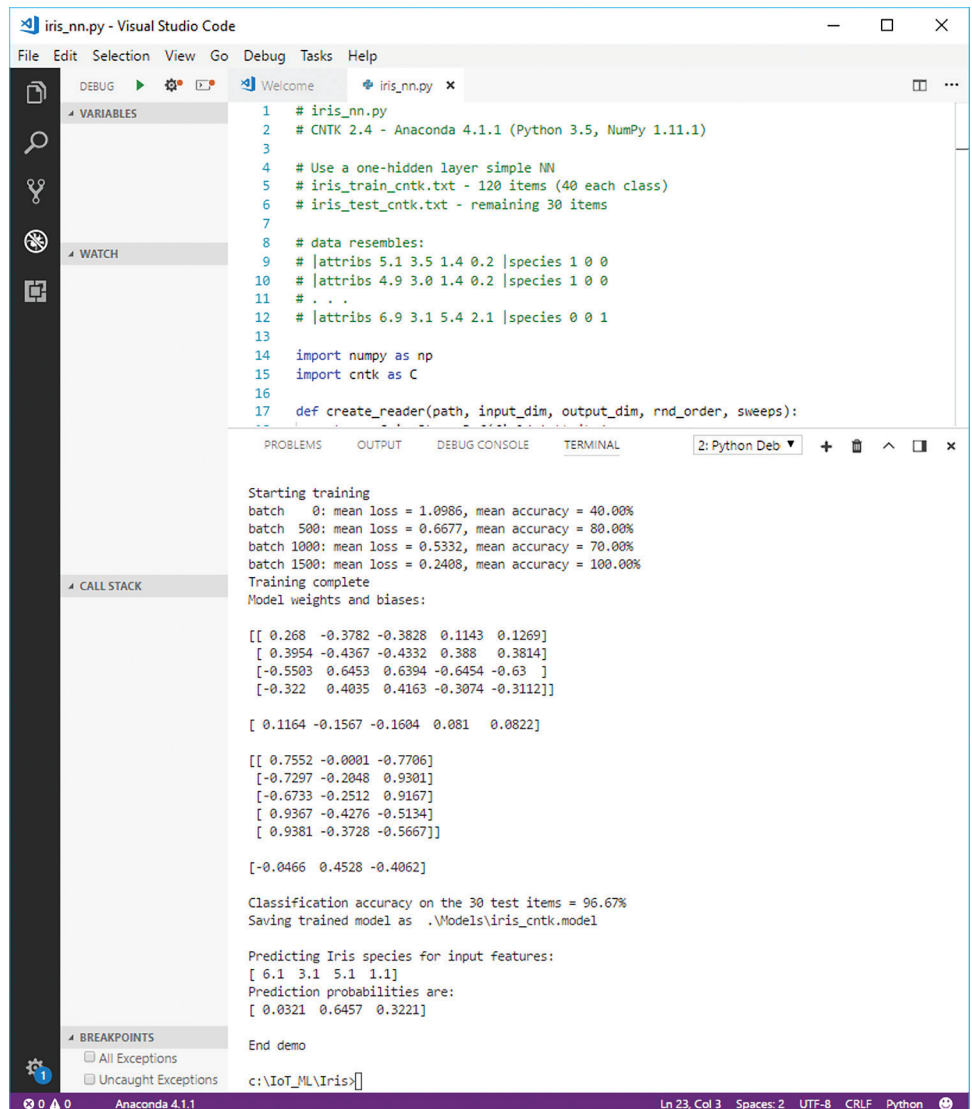


Figure 1 Creating and Training a Neural Network Model



DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

All trademarks or registered trademarks are property of their respective owners.

To summarize, an ML model is all the information needed to accept input data and generate an output prediction. In the case of a neural network, this information consists of the number of input, hidden and output nodes, the values of the weights and biases, and the types of activation functions used on the hidden and output layer nodes.

OK, but where do the values of the weights and the biases come from? They're determined by training the model. Training is using a set of data that has known input values and known, correct output values, and applying an optimization algorithm such as back-propagation to minimize the difference between computed output values and known, correct output values.

There are many other kinds of ML models, such as decision trees and naive Bayes, but the general principles are the same. When using a neural network code library such as Microsoft CNTK or Google Keras/TensorFlow, the program that trains an ML model will save the model to disk. For example, CNTK and Keras code resembles:

```
mp = "\\Models\\iris_nn.model"
model.save(mp, format=C.ModelFormat.CNTKv2) # CNTK
```

```
model.save("\\Models\\iris_model.h5") # Keras
```

ML libraries also have functions to load a saved model. For example:

```
mp = "\\Models\\iris_nn.model"
model = C.ops.functions.Function.load(mp) # CNTK
```

```
model = load_model("\\Models\\iris_model.h5") # Keras
```

Most neural network libraries have a way to save just a model's weights and biases values to file (as opposed to the entire model).

Deploying a Standard ML Model to an IoT Device

The image in **Figure 1** shows an example of what training an ML model looks like. I used Visual Studio Code as the editor and the Python language API interface to the CNTK v2.4 library. Creating a trained ML model can take days or weeks of effort, and typically requires a lot of processing power and memory. Therefore, model training is usually performed on powerful machines,

often with one or more GPUs. Additionally, as the size and complexity of a neural network increases, the number of weights and biases increases dramatically, and so the file size of a saved model also increases greatly.

For example, the 4-5-3 iris model described in the previous section has only $(4 * 5) + 5 + (5 * 3) + 3 = 43$ weights and biases. But an image classification model with millions of input pixel values and hundreds of hidden processing nodes can have hundreds of millions, or even billions, of weights and biases. Notice that the values of all 43 weights and biases of the iris example are shown in **Figure 1**:

```
[[ 0.2680 -0.3782 -0.3828 0.1143 0.1269]
 [ 0.3954 -0.4367 -0.4332 0.3880 0.3814]
 [-0.5503 0.6453 0.6394 -0.6454 -0.6300]
 [-0.322 0.4035 0.4163 -0.3074 -0.3112]]
```

```
[ 0.1164 -0.1567 -0.1604 0.0810 0.0822]
```

```
[[ 0.7552 -0.0001 -0.7706]
 [-0.7297 -0.2048 0.9301]
 [-0.6733 -0.2512 0.9167]
 [ 0.9367 -0.4276 -0.5134]
 [ 0.9381 -0.3728 -0.5667]]
```

```
[-0.0466 0.4528 -0.4062]
```

So, suppose you have a trained ML model. You want to deploy the model to a small, weak, IoT device. The simplest solution is to install onto the IoT device the same neural network library software you used to train the model. Then you can copy the saved trained model file to the IoT device and write code to load the model and make a prediction. Easy!

Unfortunately, this approach will work only in relatively rare situations where your IoT device is quite powerful—perhaps along the lines of a desktop PC or laptop. Also, neural network libraries such as CNTK and Keras/TensorFlow were designed to train models quickly and efficiently, but in general they were not necessarily designed for optimal performance when performing input-output with a trained model. In short, the easy solution for deploying a trained ML model to an IoT device on the edge is rarely feasible.

The Custom Code Solution

Based on my experience and conversations with colleagues, the most common way to deploy a trained ML model to an IoT device on the edge is to write custom C/C++ code on the device. The idea is that C/C++ is almost universally available on IoT devices, and C/C++ is typically fast and compact. The demo program in **Figure 3** illustrates the concept.

The demo program starts by using the gcc C/C++ tool to compile file test.c into an executable on the target device. Here, the target device is just my desktop PC but there are C/C++ compilers for almost every kind of IoT/CPU device. When run, the demo program displays the values of the weights and biases of the iris flower example, then uses input values of (6.1, 3.1, 5.1, 1.1) and computes and displays the output values (0.0321, 0.6458, 0.3221). If you compare **Figure 3** with **Figures 1** and **2**, you'll see the inputs, weights and biases, and outputs are the same (subject to rounding error).

Demo program test.c implements only the neural network input-output process. The program starts by setting up a struct data structure to hold the number of nodes in each layer, values for the hidden and output layer nodes, and values of the weights and biases:

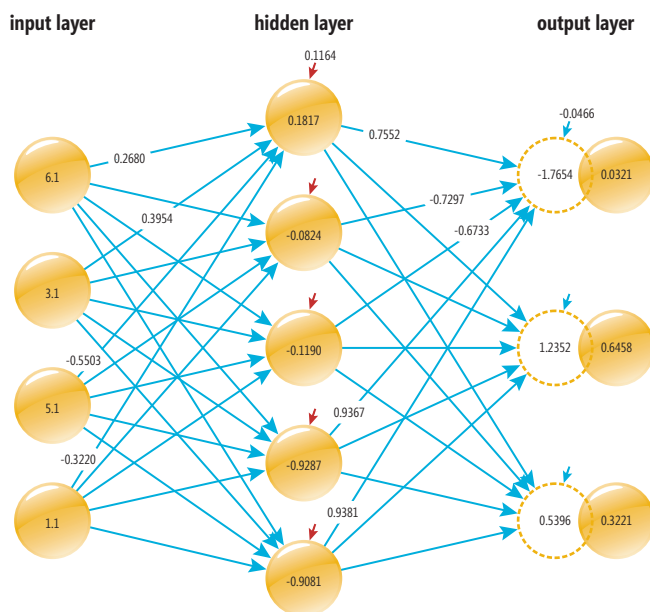


Figure 2 The Neural Network Input-Output Mechanism


```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // Has tanh()

typedef struct {
    int ni, nh, no;
    float *h_nodes, *o_nodes; // No i_nodes
    float **ih_wts, **ho_wts;
    float *h_biases, *o_biases;
} nn_t;
```

The program defines the following functions:

- construct(): initialize the struct
- free(): deallocate memory when done
- set_weights(): assign values to weights and biases
- softmax(): the softmax function
- predict(): implements the NN IO mechanism
- show_weights(): a display helper

The main advantage of using a custom C/C++ IO function is conceptual simplicity.

The key lines of code in the demo program main function look like:

```
nn_t net; // Neural net struct
construct(&net, 4, 5, 3); // Instantiate the NN

float wts[43] = { // specify the weights and biases
    0.2680, -0.3782, -0.3828, 0.1143, 0.1269,
    ...
    -0.0466, 0.4528, -0.4062 };

set_weights(&net, wts); // Copy values into NN
float inpts[4] = { 6.1, 3.1, 5.1, 1.1 }; // Inputs
int shownodes = 0; // Don't show
float* probs = predict(net, inpts, shownodes);
```

The point is that if you know exactly how a simple neural network ML model works, the IO process isn't magic. You can implement basic IO quite easily.

The main advantage of using a custom C/C++ IO function is conceptual simplicity. Also, because you're coding at a very low level (really just one level of abstraction above assembly language), the generated executable code will typically be very small and run very fast. Additionally, because you have full control over your IO code, you can use all kinds of tricks to speed up performance or reduce memory footprint. For example, program test.c uses type float but, depending on the problem scenario, you might be able to use a custom 16-bit fixed-point data type.

The main disadvantage of using a custom C/C++ IO approach is that the technique becomes increasingly difficult as the complexity of the trained ML model increases. For example, an IO function for a single hidden layer neural network with tanh and softmax

```
C:\WINDOWS\system32\cmd.exe

C:\IoT_ML\Iris>gcc -o test.exe test.c

C:\IoT_ML\Iris>test.exe

Neural net on simulated IoT device with C demo

neural net:
4-5-3

weights and biases:
0.2680 -0.3782 -0.3828 0.1143 0.1269
0.3954 -0.4367 -0.4332 0.3880 0.3814
-0.5503 0.6453 0.6394 -0.6454 -0.6300
-0.3220 0.4035 0.4163 -0.3074 -0.3112

0.1164 -0.1567 -0.1604 0.0810 0.0822

0.7552 -0.0001 -0.7706
-0.7297 -0.2048 0.9301
-0.6733 -0.2512 0.9167
0.9367 -0.4276 -0.5134
0.9381 -0.3728 -0.5667

-0.0466 0.4528 -0.4062

input:
6.1 3.1 5.1 1.1

output:
0.0321 0.6458 0.3221

C:\IoT_ML\Iris>
```

Figure 3 Simulation of Custom C/C++ IO Code on an IoT Device

activation is very easy to implement—taking only about one day to one week of development effort, depending on many factors, of course. A deep neural network with several hidden layers is somewhat easy to deal with—maybe a week or two of effort. But implementing the IO functionality of a convolutional neural network (CNN) or a long, short-term memory (LSTM) recurrent

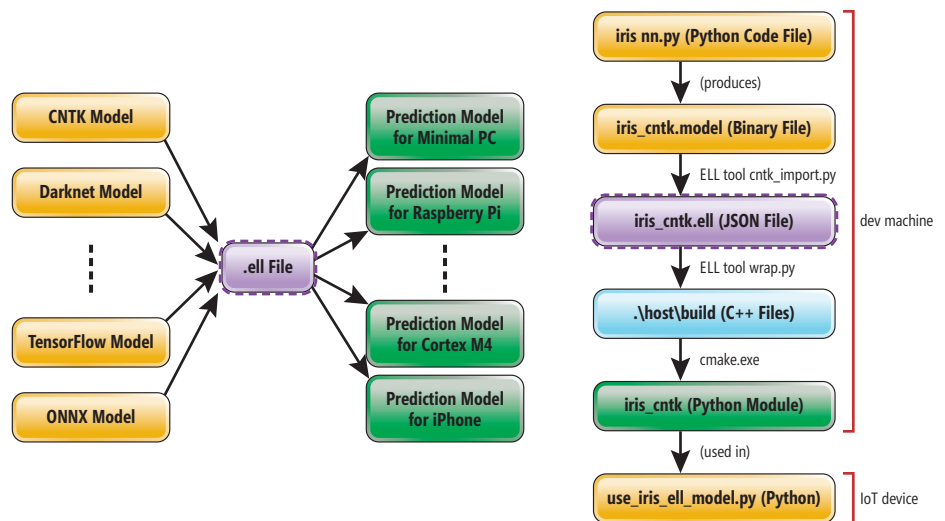


Figure 4 The ELL Workflow Process, High-Level and Granular

```
C:\WINDOWS\system32\cmd.exe

(py36) C:\IoT_ML>copy C:\ELL\docs\tutorials\shared\tutorial_helpers.py .
1 file(s) copied.

(py36) C:\IoT_ML>python use_iris_ell_model.py

Begin use ELL model demo

Input to ELL model:
[[6.1 3.1 5.1 1.1]]

Prediction probabilities:
[0.0321 0.6457 0.3221]

End ELL demo

(py36) C:\IoT_ML>
```

Figure 5 Simulation of Using an ELL Model on an IoT Device

neural network is very difficult and would typically require much more than four weeks of development effort.

I suspect that as the use of IoT devices increases, there will be efforts to create open source C/C++ libraries that implement the IO for ML models created by different neural network libraries such as CNTK and Keras/TensorFlow. Or, if there's enough demand, the developers of neural network libraries might create C/C++ IO APIs for IoT devices themselves. If you had such a library, writing custom IO for an IoT device would be relatively simple.

The Microsoft Embedded Learning Library

The Microsoft Embedded Learning Library (ELL) is an ambitious open source project intended to ease the development effort required to deploy an ML model to an IoT device on the edge (microsoft.github.io/ELL). The basic idea of ELL is illustrated on the left side of **Figure 4**.

In words, the ELL system accepts an ML model created by a supported library, such as CNTK, or a supported model format, such as open neural network exchange (ONNX). The ELL system uses the input ML model and generates an intermediate model as an .ell file. Then the ELL system uses the intermediate .ell model file to generate executable code of some kind for a supported target device. Put another way, you can think of ELL as a sort of cross-compiler for ML models.

A more granular explanation of how ELL works is shown on the right side of **Figure 4**, using the iris flower model example. The process starts with an ML developer writing a Python program named `iris_nn.py` to create and save a prediction model named `iris_cntk.model`, which is in a proprietary binary format. This process is shown in **Figure 1**.

The ELL command-line tool `cntk_import.py` is then used to create an intermediate `iris_cntk.ell` file, which is stored in JSON format. Next, the ELL command-line tool `wrap.py` is used to generate a directory `host\build` of C/C++ source code files. Note that “host” means to take the settings from the current machine, so a more common scenario would be something like `\pi3\build`. Then the `cmake.exe` C/C++ compiler-build tool is used to generate

a Python module of executable code, containing the logic of the original ML model, named `iris_cntk`. The target could be a C/C++ executable or a C# executable or whatever is best-suited for the target IoT device.

The `iris_cntk` Python module can then be imported by a Python program (`use_iris_ell_model.py`) on the target device (my desktop PC), as shown in **Figure 5**. Notice that the input values (6.1, 3.1, 5.1, 1.1) and output values (0.0321, 0.6457, 0.3221) generated by the ELL system model are the same as the values generated during model development (**Figure 1**) and the values generated by the custom C/C++ IO function (**Figure 3**).

The leading “(py36)” before the command prompts in **Figure 5** indicate I’m working in a special Python setting called a Conda environment where I’m using Python version 3.6, which was required at the time I coded my ELL demo.

The code for program `use_iris_ell_model.py` is shown in **Figure 6**. The point is that ELL has generated a Python module/package that can be used just like any other package/module.

The ELL system is still in the very early stages of development, but based on my experience, the system is ready for you to experiment with and is stable enough for limited production development scenarios.

I expect your reaction to the diagram of the ELL process in **Figure 4** and its explanation is something like, “Wow, that’s a lot of steps!” At least, that was my reaction. Eventually, I expect the ELL system to mature to a point where you can generate a model for deployment to an IoT device along the lines of:

```
source_model = ".\\iris_cntk.model"
target_model = ".\\iris_cortex_m4.model"
ell_generate(source_model, target_model)
```

But for now, if you want to explore ELL you’ll have to work with several steps. Luckily, the ELL tutorial from the ELL Web site on which much of this article is based is very good. I should point out that to get started with ELL you must install ELL on your

Figure 6 Using an ELL Model in a Python Program

```
# use_iris_ell_model.py
# Python 3.6

import numpy as np
import tutorial_helpers # used to find package
import iris_cntk as m    # the ELL module/package

print("\nBegin use ELL model demo \n")

unknown = np.array([[6.1, 3.1, 5.1, 1.1]],
                    dtype=np.float32)

np.set_printoptions(precision=4, suppress=True)
print("Input to ELL model: ")
print(unknown)
predicted = m.predict(unknown)
print("\nPrediction probabilities: ")
print(predicted)
print("\nEnd ELL demo \n")
```

desktop machine, and installation consists of building C/C++ source code—there's no .msi installer for ELL (yet).

A cool feature of ELL that isn't obvious is that it performs some very sophisticated optimization behind the scenes. For example, the ELL team has explored ways to compress large ML models, including sparsification and pruning techniques, and replacing floating point math with 1-bit math. The ELL team is also looking at algorithms that can be used in place of neural networks, including improved decision trees and k-DNF classifiers.

The tutorials on the ELL Web site are quite good, but because there are many steps involved, they are a bit long. Let me briefly sketch out the process so you can get a feel for what installing and using ELL is like. Note that my commands are not syntactically correct; they're highly simplified to keep the main ideas clear.

Installing the ELL system resembles:

```
> (install several tools such as cmake and BLAS)
> git clone https://github.com/Microsoft/ELL.git
> cd ELL
> nuget.exe restore external/packages.config -PackagesDirectory external
> md build
> cd build
> cmake -G "Visual Studio 15 2017 Win64" ..
> cmake --build . --config Release
> cmake --build . --target _ELL_python --config Release
```

In words, you must have quite a few tools installed before starting, then you pull the ELL source code down from GitHub and then build the ELL executable tools and Python binding using cmake.

Creating an ELL model resembles:

```
> python cntk_import.py iris_cntk.model
> python wrap.py iris_nn_cntk.ell --language python --target host
> cd host
> md build
> cd build
> cmake -G "Visual Studio 15 2017 Win64" .. && cmake --build . --config release
```

That is, you use ELL tool cntk_import.py to create a .ell file from a CNTK model file. You use wrap.py to generate a lot of C/C++ specific to a particular target IoT device. And you use cmake to generate executables that encapsulate the original trained ML models behavior.

Wrapping Up

To summarize, a machine learning model is all the information needed for a software system to accept input and generate a prediction. Because IoT devices on the edge often require very fast and reliable performance, it's sometimes necessary to compute ML predictions directly on a device. However, IoT devices are often small and weak, so you can't simply copy a model that was developed on a powerful desktop machine to the device. A standard approach is to write custom C/C++ code, but this approach doesn't scale to complex ML models. An emerging approach is the use of ML cross-compilers, such as the Microsoft Embedded Learning Library.

When fully mature and released, the ELL system will quite likely make developing complex ML models for IoT devices on the edge dramatically easier than it is today. ■

Dr. James McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.

THANKS to the following Microsoft technical experts who reviewed this article: Byron Changuion, Chuck Jacobs, Chris Lee and Ricky Loynd



Instantly Search Terabytes

dtSearch's **document filters** support:

- popular file types
- emails with multilevel attachments
- a wide variety of databases
- web data

Over 25 search options including:

- efficient multithreaded search
- **easy** **multicolor** **hit-highlighting**
- forensics options like credit card search

Developers:

- APIs for .NET, C++ and Java; ask about new cross-platform .NET Standard SDK with Xamarin and .NET Core
- SDKs for Windows, UWP, Linux, Mac, iOS in beta, Android in beta
- FAQs on faceted search, granular data classification, Azure and more

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval®
since 1991

dtSearch.com 1-800-IT-FINDS

Improving LUIS Intent Classifications

Zvi Topol

The Language Understanding Intelligence Service (LUIS), which is part of Microsoft Cognitive Services, offers a machine learning solution for natural language understanding. There are many use cases for LUIS, including chat bots, voice interfaces and cognitive search engines.

In a nutshell, when given a textual user input, also known as an *utterance*, LUIS returns the *intent* detected behind the utterance, that is, what the user intends to ask about. It also detects the different *entities*—references to real-world objects—that appear in the utterance. Additionally, it outputs a confidence score for each intent and entity detected. Those are numbers in the range [0, 1], with 1 indicating the most confidence about the detection and 0 being the least confident about it.

Previous *MSDN Magazine* articles have covered the basics of LUIS in detail. In particular, I encourage you to refer to the article, “Enable Natural Language Interaction with LUIS,” by Ashish Sahu (msdn.com/magazine/mt745095) for additional information about how to get started with LUIS.

This article discusses:

- LUIS intent classification
- Visualizing how utterances fit into your model’s intents using Scattertext
- Explaining intent classification using LIME

Technologies discussed:

LUIS, QnA Maker, Scattertext, Python, LIME

This article will focus on two open source tools, Scattertext and LIME, which can help you understand the detection and classification of intents by LUIS. (In what follows, I’ll use detection and classification interchangeably.)

In particular, I’ll show how such tools can be used to shed some light on the classification process and explain why LUIS is uncertain about its intent detection in some cases—typically situations in which the top intents detected for a given utterance have similar confidence scores, for example a 50-50 split between two intents. It’s more likely to output the wrong intent in such situations.

While LUIS currently supports some troubleshooting capabilities, including active learning to help identify and retrain utterances it’s uncertain about, there are no word-level visualization and analysis tools that can further help resolve such uncertainty. Scattertext and LIME can help in overcoming that limitation.

Now let’s take a look at a simple FinTech case that will serve as a running example. Imagine you work for a bank and you’ve been tasked with understanding user questions that fall into two categories:

- Questions about their personal bank accounts, such as:
 - “What is my savings account balance?”
 - “What is the latest transaction in my checking account?”
 - “I would like my savings statement to be sent again”
 - “Have I received my April salary yet?”
 - “When was the last cell phone auto pay processed?”
 - “What are annual rates for my savings accounts?”
 - “What is the balance in my checking account?”
- Questions or requests about other banking services, including mortgages, auto loans, and so forth, such as:

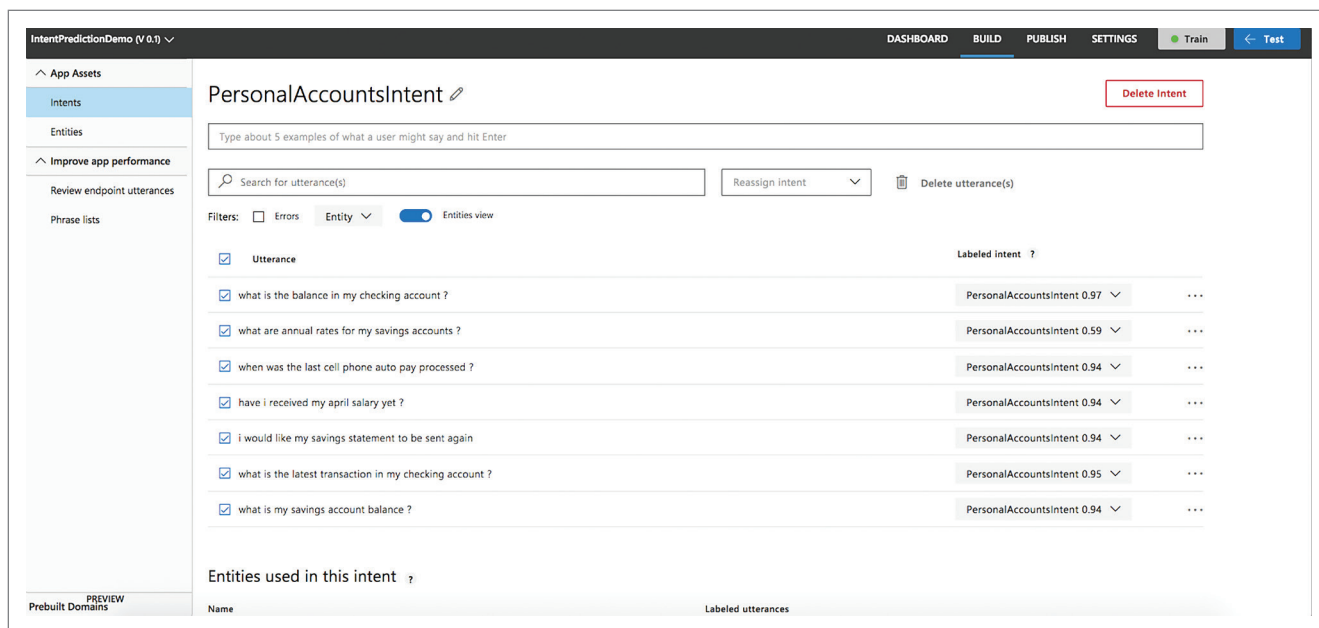


Figure 1 PersonalAccountsIntent Utterances with Their Confidence Scores

“I would like to get assistance about mortgage rates”
 “Whom can I speak with regarding mortgages?”
 “What is the annual rate for the one-year savings account?”
 “What terms do you offer for mortgages?”
 “Who is responsible for mortgages?”
 “What are annual rates for savings accounts?”
 “How are your mortgage rates compared to other banks?”

The plan is to use LUIS for natural language understanding of the user requests. One way to go about this is to create two intents and train LUIS to detect them.

Let’s call the first category’s intent PersonalAccountsIntent, and the second category’s intent OtherServicesIntent. You can then use the utterance examples previously listed to train LUIS. It will create a third “catch-all” intent automatically called None for general utterances, which should be very different from the first two intents. You can also provide additional examples for the None intent.

Ideally, you want your intents to be distinguishable from one another with a high degree of certainty.

After training, you can publish your model to production. You can also see the utterances along with the confidence scores for the different intents in the LUIS UI, as shown in **Figure 1**.

The dashboard offers some basic summary statistics about the application. If you look at the dashboard in **Figure 1**, you’ll notice that the lowest confidence score for PersonalAccountsIntent is 0.59 and is obtained for the utterance, “what are annual rates for my savings accounts?” The confidence score for this utterance to be classified as OtherServicesIntent is pretty close at 0.44. This means that LUIS is not very certain as to how to classify this intent.

Ideally, you want your intents to be distinguishable from one another with a high degree of certainty, that is, to have one intent with a very high confidence score, while other intents have very low scores. If you revisit the utterance lists for both intents, you’ll see there’s another very similar utterance example (“what is the annual rate for the one-year savings account?”) that’s labeled differently as OtherServicesIntent.

Using this insight, you can fine-tune your utterance samples to use different and distinct words.

Here, I’ve presented seven utterance examples for each intent. But what if there were multiple intents (at the time of writing LUIS can classify up to 500 different intents) and many more utterance examples for each intent?

Clearly, a more systematic approach is needed to address such a challenge. In what follows, I’ll show how Scattertext and LIME can help.

Understanding Intent Classification Using Scattertext

Scattertext is an open source tool written in Python by Jason Kessler. You’ll find the source code and a tutorial at bit.ly/2G0DLmp, and a paper entitled “Scattertext: a Browser-Based Tool for Visualizing How Corpora Differ,” which explains the tool in detail, at bit.ly/2G05ow6.

Scattertext was conceived as a tool to visualize the differences and similarities between two collections of text articles, also known as corpora, and has various features you may find useful; for example, it also supports emojis.

In this article, I’m going to leverage the tool to produce a visualization of the differences and similarities between the utterance examples for the two intents, PersonalAccountsIntent and OtherServicesIntent.

To install Scattertext, which requires Python version 3, follow the installation instructions in the tutorial. I also recommended you install Spacy, an open source Natural Language Processing library (spacy.io) and Pandas (pandas.pydata.org), another open source library that lets you work with tabular data in-memory.

Figure 2 Code for Scattertext Visualization

```
import scattertext as st
import space
import pandas as pd

examples_data_location = 'example.csv'

two_df = pd.read_csv(examples_data_location, encoding = 'utf8')

nlp = spacy.en.English()

corpus = st.CorpusFromPandas(two_df,
                             category_col='intent',
                             text_col='utterance',
                             nlp=nlp).build()

html = st.produce_scattertext_explorer(corpus,
                                     category='PersonalAccountsIntent', category_name='PersonalAccountsIntent',
                                     not_category_name='OtherServicesIntent', width_in_pixels=1000)
open("MSDN-Visualization.html", 'wb').write(html.encode('utf-8'))
```

Now I need to feed the utterance examples into Scattertext. To do that, I'll create a CSV table with two columns, one for the utterances and the other for the intents. The utterance column will include the utterance examples as one string, separated by the new-line character. (If you're using Excel, you can use Alt+Enter to enter multiple lines into a single cell.) The intent column will include the labels of the intents, in this case, PersonalAccountsIntent and OtherServicesIntent. So, for this example the result is a 2x2 CSV table.

You can now use Python to run the code in **Figure 2**. The code will load the CSV table into a Panda data frame and then hand it over to Scattertext, specifying a few parameters related to categories (the intents) and the output format.

Scattertext will produce an HTML page that includes a visualization showing the top words unique for each intent, as well as those shared by both intents. There's also a search box that lets you look for particular words, that if found, are highlighted in the

visualization. In a crowded visualization, this can be very useful.

Figure 3 shows the Scattertext output for this example.

Scattertext works by counting word frequencies for each intent's utterance examples and displaying the words in a way that makes it easier to determine differences and similarities between the intents. At this point, the counts only include one-word expressions (unigrams). However, if you have expressions that include multiple words, such as "auto pay," you can do some pre-processing to specify what you want. For example, you could represent "auto pay" as "auto_pay."

The visualization in **Figure 3** shows the two intents—OtherServicesIntent on the X axis and PersonalAccountsIntent on the Y axis. Words that appear closer to the bottom right are more likely to appear in utterance examples for OtherServicesIntent, such as "mortgages" and "rates," while words that appear on the top left are those that are more likely to appear in utterance examples for PersonalAccountsIntent, such as "my" and "account." Words on the diagonal are likely to appear in utterance examples for both intents, for example, "savings" or "what."

Learning that certain words appear frequently in both intents' utterance examples can help you fine-tune the utterance examples to improve classification confidence and accuracy.

One way to do so is by adding more distinct words or by even rephrasing each intent's utterance examples that include the words frequently in both so as to render them more distinguishable.

The advantage of using Scattertext is that it's possible to get value from the tool even for small data sets, such as my toy example with only seven utterance examples for each intent. Clearly, the more utterance examples per intent you have, the more complicated it becomes to find the differences and similarities among them. Scattertext can help you appreciate the differences and similarities in a rapid visual way.

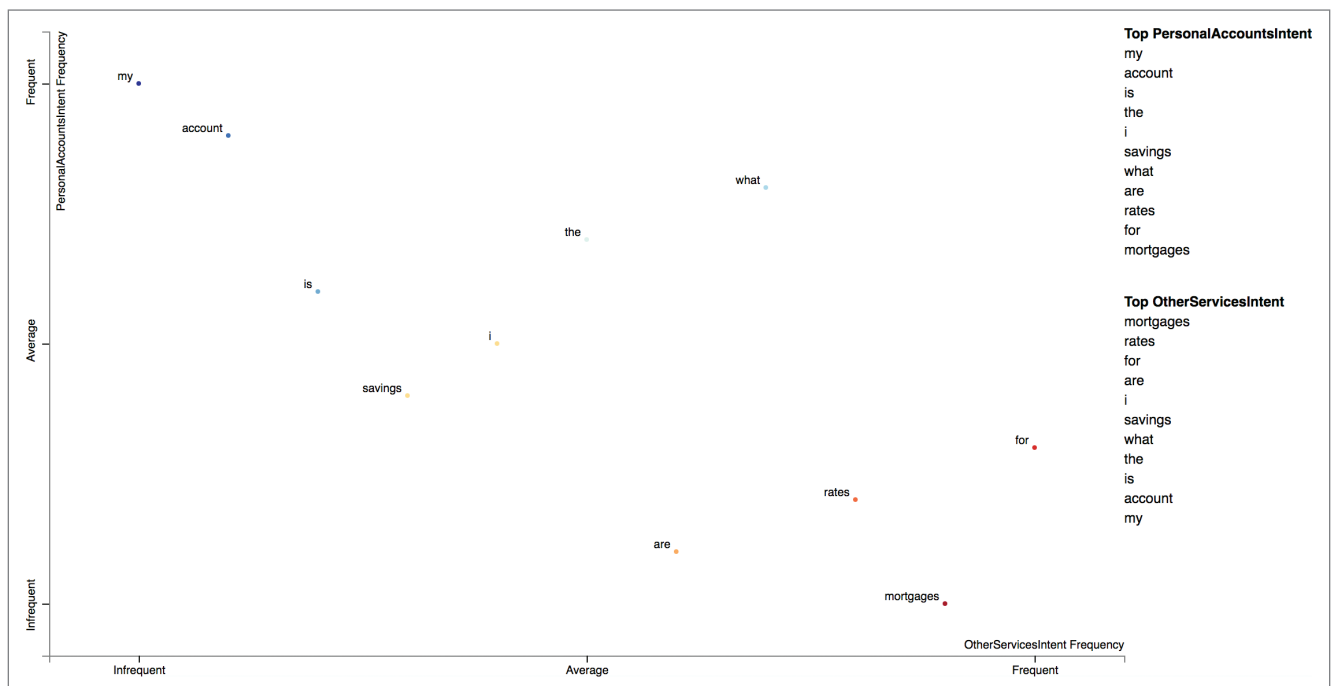


Figure 3 Scattertext Visualization



LEADTOOLS Document Imaging SDKs V19

from \$2,995.00 SRP

LEADTOOLS®
THE WORLD LEADER IN IMAGING SDKs

Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services



DevExpress DXperience 18.1

from \$1,439.99

 **DevExpress**

The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics



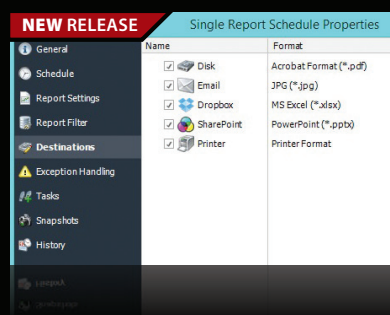
Help & Manual Professional

from \$586.04

 **ec software**

Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePub, Kindle or print
- Styles and Templates give you full design control



PBRS (Power BI Reports Scheduler)

from \$8,132.21

christianstevenson

Data Driven Distribution for Power BI Reports & Dashboards.

- A comprehensive set of job (schedule) types gives you the power to automate delivery in Power BI
- Automate report delivery & send reports to printer, fax, folder, FTP, DropBox, SharePoint & email
- Contains powerful system event triggered, data-driven and business process workflow functions
- Respond instantly by firing off reports and automation scripts when an event occurs

Figure 4 Using LIME to Analyze Utterances

```
import requests
import json
from lime.lime_text import LimeTextExplainer
import numpy as np

def call_with_utterance_list(utterance_list):

    scores=np.array([call_with_utterance(utterance) for utterance in
        utterance_list])

    return scores

def call_with_utterance(utterance):

    if utterance is None:
        return np.array([0, 1])

    app_url ='your_url_here&q='
    r = requests.get(app_url+utterance)

    json_payload = json.loads(r.text)

    intents = json_payload['intents']

    personal_accounts_intent_score =
        [intent['score'] for intent in intents if intent['intent'] ==
            'PersonalAccountsIntent']
    other_services_intent_score = [intent['score'] for intent in intents if
        intent['intent'] == 'OtherServicesIntent']
    none_intent_score = [intent['score'] for intent in intents if
        intent['intent'] == 'None']

    if len(personal_accounts_intent_score) == 0:
        return np.array([0, 1])

    normalized_score_denom = personal_accounts_intent_score[0]+
        other_services_intent_score[0]+none_intent_score[0]

    score = personal_accounts_intent_score[0]/normalized_score_denom

    complement = 1 - score

    return (np.array([score, complement]))

if __name__ == "__main__":

    explainer = LimeTextExplainer(class_names=['PersonalAcctIntent', 'Others'])
    utterance_to_explain = 'What are annual rates for my savings accounts'
    exp = explainer.explain_instance(utterance_to_explain,
        call_with_utterance_list, num_samples=500)
    exp.save_to_file('lime_output.html')
```

It's also worth noting that you can use Scattertext in a similar fashion when you have more than two intents by comparing pairs of intents at a time.

Explaining Intent Classifications Using LIME

Now let's look at an open source tool called LIME, or Local Interpretable Model-Agnostic Explanation, which allows you to explain intent classification. You'll find the source code and a tutorial at bit.ly/2l4Mp9z, and an academic research paper entitled, "Why Should I Trust You?: Explaining the Predictions of Any Classifier" (bit.ly/2ocHXKv).

LIME is written in Python and you can follow the installation instructions in the tutorial before running the code in **Figure 4**.

LIME allows you to explain classifiers for different modalities, including images and text. I'm going to use the text version of LIME, which outputs word-level insights about the various words in the utterance. While I'm using LUIS as my classifier of choice, a wide range of classifiers can be fed into LIME; they're essentially treated as black boxes.

The text version of LIME works roughly as follows: It randomly creates multiple modifications or samples of the input utterance

by removing any number of words, then calls LUIS on each one of them. The number of samples is controlled by the parameter `num_samples`, which in **Figure 4** is set to 500. For the example utterance, modified utterances can include variations such as "are annual for accounts" and "what annual rates for my savings."

LIME allows you to explain classifiers for different modalities, including images and text.

LIME uses the confidence scores returned from LUIS to fit a linear model that then estimates the effects of single words on classification confidence scores. This estimation helps you identify how the confidence score is likely to change if you were to remove words from the utterance and run the classifier again (as I show later).

The only major requirement for the classifier is to output confidence scores for the classified labels. Confidence scores over the

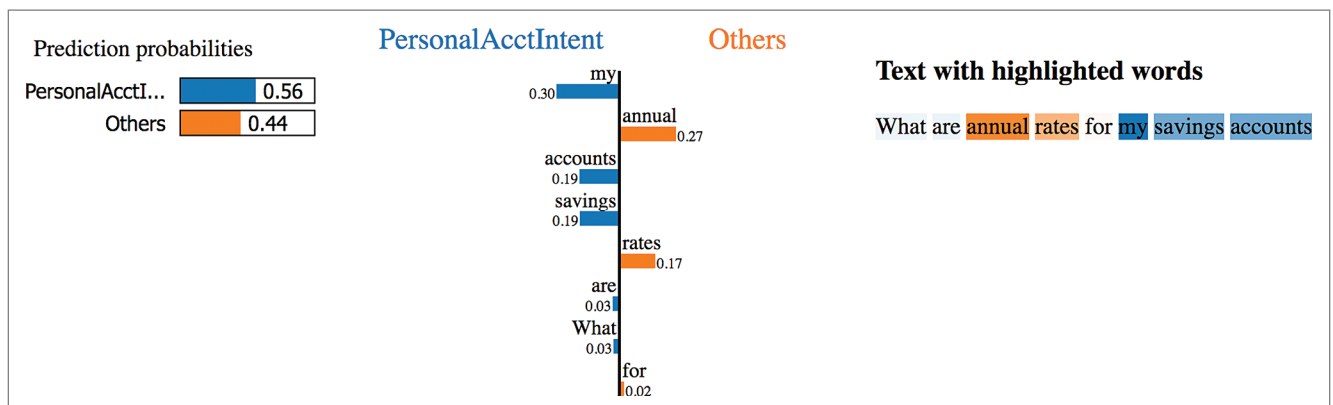


Figure 5 LIME Output for the "What Are Annual Rates for My Savings Accounts?" Utterance

Figure 6 Results for the “What Are Annual Rates for My Savings Accounts?” Query

```
{
  "query": "what are annual rates for savings accounts",
  "topScoringIntent": {
    "intent": "OtherServicesIntent",
    "score": 0.577525139
  },
  "intents": [
    {
      "intent": "OtherServicesIntent",
      "score": 0.577525139
    },
    {
      "intent": "PersonalAccountsIntent",
      "score": 0.267547846
    },
    {
      "intent": "None",
      "score": 0.00754897855
    }
  ],
  "entities": []
}
```

different categories are treated as a probability distribution, and therefore should be in the range of [0,1] and sum to 1. LUIS outputs confidence scores in that range for the defined intents and the additional None intent, but those aren't guaranteed to sum to 1. Therefore, when using LIME, I'll normalize the LUIS scores to sum to 1. (This is done in the function call `_with_utterance`.)

The code listed in **Figure 4** uses LIME to produce an explanation about the prediction for the utterance, “what are annual rates for my savings accounts?” It then generates an HTML visualization, which is presented in **Figure 5**.

In **Figure 5** you can see the predicted probabilities for the utterance, focused here on `PersonalAccountsIntent` rather than the two other intents, `OtherServicesIntent` and `None`. (Note that the probabilities are very close to but not exactly the same as the confidence scores output by LUIS due to normalization.) You can also see the most significant words for classifying the intent as `PersonalAccountsIntent` (those are words on top of the blue bars and are also highlighted in blue in the utterance text). The weight of the bar indicates the effect on the classification confidence score should the word be removed from the utterance. So, for example, “my” is the word with the most significant effect for detecting the utterance's intent in this case. If I were to remove it from the utterance, the confidence score would be expected to reduce by 0.30, from 0.56 to 0.26. This is an estimation generated by LIME. In fact, when removing the word and feeding the “what are annual rates for savings accounts?” utterance into LUIS, the result is that the confidence score for `PersonalAccountsIntent` is 0.26 and the intent is now classified as `OtherServicesIntent`, with a confidence score of about 0.577 (see **Figure 6**).

Other significant words are “accounts” and “savings,” which together with “my” provide similar insights to the ones provided by Scattertext.

Two important words with significant negative weights are “annual” and “rates.” This means that removing them from the utterance would increase the confidence scores for the utterance to be classified as `PersonalAccountsIntent`. Scattertext showed that “rates” is more common in utterance examples for `OtherServicesIntent`, so this isn't a big surprise.

Figure 7 Results for the “What Are Rates for My Savings Accounts?” Query

```
{
  "query": "what are rates for my savings accounts",
  "topScoringIntent": {
    "intent": "PersonalAccountsIntent",
    "score": 0.71332705
  },
  "intents": [
    {
      "intent": "PersonalAccountsIntent",
      "score": 0.71332705
    },
    {
      "intent": "OtherServicesIntent",
      "score": 0.18973498
    },
    {
      "intent": "None",
      "score": 0.007595492
    }
  ],
  "entities": []
}
```

However, there is something new to be learned from LIME—the word “annual” is significant for LUIS in determining that the intent in this case doesn't belong in the `PersonalAccountsIntent`, and removing it is expected to increase the confidence score for `PersonalAccountsIntent` by 0.27. Indeed, when I remove annual before feeding the utterance, I get a higher confidence score for the `PersonalAccountsIntent` intent, namely 0.71 (see **Figure 7**).

In this way, LIME helps you identify significant words that drive classification confidence scores. It can thus provide insights that help you fine-tune your utterance examples to improve intent classification accuracy.

Wrapping Up

I have shown that when developing an application based on NLU, intent prediction for some utterances can be rather challenging and can be helped by a better understanding of how to fine-tune utterance examples in order to improve classification accuracy.

The task of understanding word-level differences and similarities among utterances can yield concrete guidance in the fine-tuning process.

I've presented two open source tools, Scattertext and LIME, that provide word-level guidance by identifying significant words that affect intent prediction. Scattertext visualizes differences and similarities of word frequencies in utterance examples, while LIME identifies significant words affecting intent classification confidence scores.

I hope these tools will help you build better NLU-based products using LUIS. ■

Zvi Topol has been working as a data scientist in various industry verticals, including marketing analytics, media and entertainment and Industrial Internet of Things. He has delivered and lead multiple machine learning and analytics projects including natural language and voice interfaces, cognitive search, video analysis, recommender systems and marketing decision support systems. He can be contacted at zvitol@gmail.com.

THANKS to the following Microsoft technical expert who reviewed this article:
Ashish Sahu

TEXTCONTROL

TX Text Control X15

Automate your reports and create beautiful documents in Windows Forms, WPF, ASP.NET and Cloud applications.

Text Control Reporting combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users create documents and templates using ordinary Microsoft Word skills.

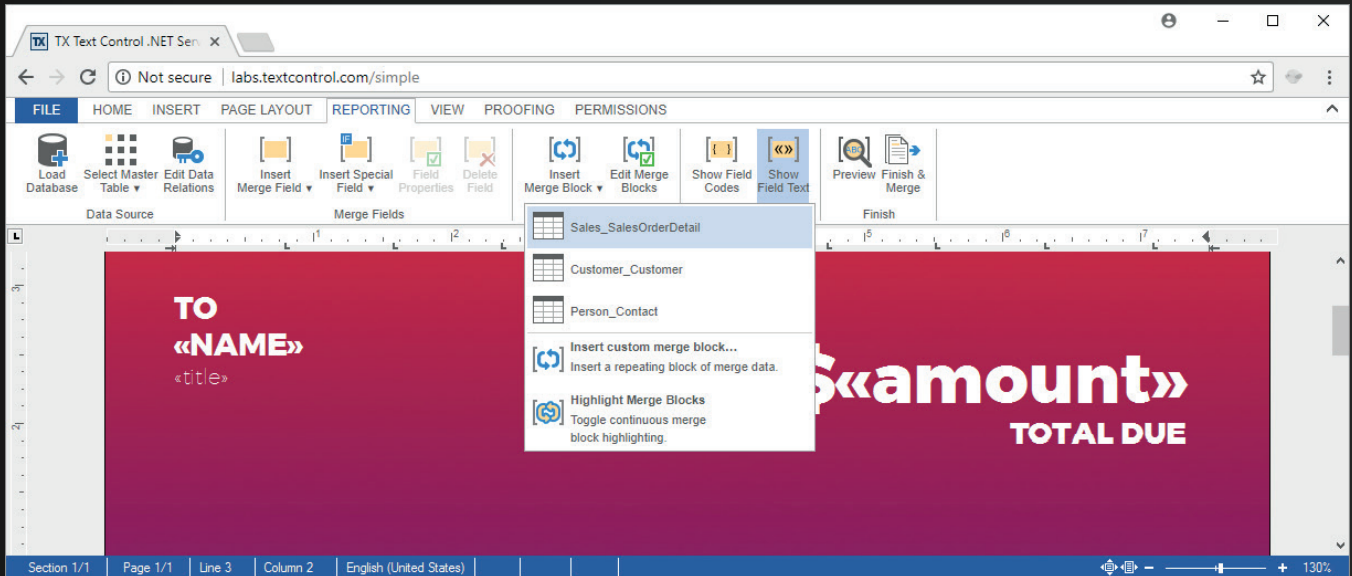
Download a free trial at
www.textcontrol.com



**WE ARE CHANGING
THE WAY YOU LOOK AT
REPORTING**

TX Text Control .NET Server for ASP.NET

Complete reporting and word processing for ASP.NET Web Forms and MVC



✓ Give your users a WYSIWYG, MS Word compatible, HTML5-based, cross-browser editor to create powerful reporting templates and documents anywhere.

✓ Text Control Reporting combines the power of a reporting tool and an easy-to-use WYSIWYG word processor - fully programmable and embeddable in your application.

✓ Replacing MS Office Automation is one of the most typical use cases. Automate, edit and create documents with Text Control UI and non-UI components.



WINDOWS FORMS



WPF



ASP.NET MVC



ASP.NET AJAX



CLOUD WEB API

TEXTCONTROL

Decentralized Applications with Azure Blockchain as a Service

Stefano Tempesta

Blockchain has captured the attention of the business and technology world as a way to streamline business processes, verify transactions, and reduce the potential for fraud. This article introduces Blockchain as a Service (BaaS) in Microsoft Azure, showing how it can be used to build a secured data structure and create a distributed transactional digital ledger.

There's plenty of literature on the Internet about blockchain and how it started as a digital ledger for Bitcoin. A good introductory article to what blockchain is can be found at bit.ly/2Is0WeJ, and for a more technical overview on how blockchain works, please refer to Jonathan Waldman's "Blockchain Fundamentals" article in the March 2018 issue of *MSDN Magazine* (msdn.com/magazine/mt845650).

Blockchain is a secure, shared, distributed ledger that can be public, private or consortium (that is, restricted to named members

only). It's secure because it uses cryptography to create transactions that are difficult (if not impossible with current computing technology) with which to tamper. Shared among all nodes or peers in the chain is a data store and, as you'll see shortly, business logic in the form of contracts. A blockchain value is indeed directly linked to the number of entities that participate in them. Critically, blockchain data and contracts are distributed, which means that there are many replicas of the database. And the more replicas there are, the more authentic it becomes. And finally, blockchain is a digital ledger, a transactional database that appends only immutable records of every transaction that occurs.

I'd like to reinforce this point about blockchain being a distributed ledger. Traditional ledgers are centralized and use third-party systems, or middlemen, to approve and record transactions. Think of credit cards, banks, identity management systems and the like. This approach creates a challenge of trust and scale. Do you trust your middleman agent to act as a broker for all your transactions? Can the agent become a single point of failure? Can it be compromised?

In a blockchain, ledgers are distributed across the entire network, and there's no need for any third-party system to be in the middle of a transaction. The technology maintains multiple replicas of data, as in a peer-to-peer file-sharing system, as each peer obtains a copy of the entire dataset. No one owns the entire thing, but everyone possesses a copy of it. **Figure 1** depicts this arrangement.

The first blockchain, Bitcoin, emerged in 2009, with distinct limits. As a digital ledger, it simply records transactions and doesn't keep

This article discusses:

- Microsoft offerings for blockchain development
- Deploying an Ethereum ledger on Azure
- Publishing a Solidity smart contract

Technologies discussed:

Azure Blockchain, Ethereum, Solidity

Code download available at:

bit.ly/2INGNEP

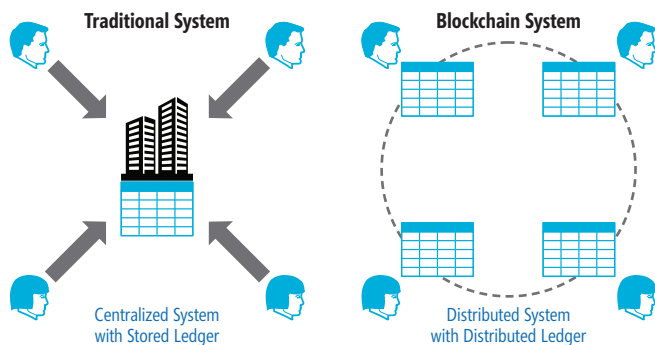


Figure 1 A Decentralized Distributed Ledger

track of digital asset account balances. Ownership of bitcoins is verified by links to previous transactions, following the immutable history of blocks in the chain of recorded transactions. Also, Bitcoin doesn't define any specific logic on how to handle a transaction and the conditions, for example, that the two involved parties must agree upon in a cryptocurrency exchange.

Blockchain technology evolved with the addition of smart contracts, which are small pieces of code that add logic to transactions. Think of smart contracts as a computer code representation of legal terms in a contract for goods or services. New blockchain ledgers emerged in the market, the most popular being Ethereum (ethereum.org) and Hyperledger Fabric (hyperledger.org), to add smart contract capability to the network. In these (let's call them Blockchain 2.0) digital ledgers, smart contracts are now stored in a block and are distributed to all nodes along with related data.

Blockchain 3.0

Bitcoin's blockchain is often referred to as Blockchain 1.0. It's a simple ledger that records transactions in sequence and represents the state of the network at any given moment. Think of it simply as a distributed database.

But just as databases have evolved over time by adding logic execution capability—in the form of stored procedures, for example—blockchain has introduced smart contracts to handle the logic tier. However, smart contracts can operate on data only contained in the block where they're stored. They can't access external data or systems, as calling a service outside of the blockchain breaks the "circle of trust" that blockchain provides for cryptographic security and immutability of transactions. CRM, ERP and payroll systems all represent external entities that aren't part of a blockchain, but may be involved in the exchange of data within a transaction. Blockchains need a way to securely receive external data, as well as access to secure execution of off-chain code.

To address this requirement, Microsoft introduced cryptlets as part of "Blockchain 3.0," the blockchain of data, logic and cloud services. **Figure 2** shows the progression of features.

Cryptlets are off-chain code modules written in any language that can execute within a secure, isolated, trusted container and can communicate over secure channels. Cryptlets extend smart contracts to the outside world by providing services like encryption, time and date events, external data access, and identity authentication. Microsoft introduced cryptlets as part of its open source

project code-named "Bletchley" (bit.ly/2lv9VZz), which has evolved into the Azure Blockchain Workbench product revealed at the Microsoft Build developer conference in May.

As described on the Web site, Bletchley is an architectural approach to building an enterprise consortium blockchain ecosystem. To be clear, this is not a blockchain stack. It's Microsoft's approach to bringing distributed ledger (blockchain) platforms into the enterprise and building real solutions addressing real business problems, while keeping the platform open.

Azure Blockchain Workbench (aka.ms/abcworkbench) leverages different blockchain ledgers and existing cloud services to enable a robust blockchain ecosystem for the enterprise. It's an easy-to-use tool with a simplified interface that enables users to create end-to-end blockchain applications that leverage the best of Azure services, including Azure Active Directory (Azure AD), Azure Key Vault, Azure SQL Database, Application Insights, Azure Functions and Service Bus. And it does so around popular blockchains and into a reference architecture that can be used to build blockchain-based applications.

Cryptlets are off-chain code modules that can execute within a secure, isolated, trusted container and can communicate over secure channels.

You can learn more about Azure Blockchain Workbench in the "Introducing Azure Blockchain Workbench" article I wrote for the June issue of *MSDN Magazine* (msdn.com/magazine/mt846726)

Getting back to cryptlets, these provide an approachable way for developers to use cross-cutting capabilities like integration into third-party systems and data access. But before I shift focus to development of decentralized applications on a blockchain, I need a platform for delivering a secure and integrated solution on public or private distributed ledgers. Microsoft Azure offers a world-wide footprint that allows building a hyper-scale, secure data and execution platform to deliver the next-generation applications on any blockchain platform.

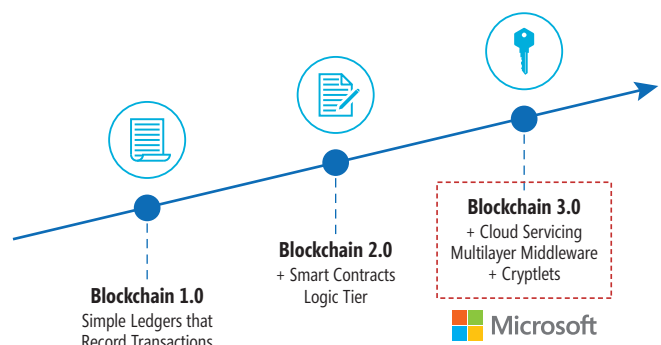


Figure 2 Evolution of Blockchain

Blockchain as a Service

Blockchain on Azure (bit.ly/2rQU05q) provides a rapid, low-cost, low-risk platform for building and deploying blockchain applications. Azure, basically, offers Blockchain as a Service (BaaS), by providing several easy-to-deploy, enterprise-ready templates for the most popular ledgers, including Ethereum, Quorum, Hyperledger Fabric, Corda and more.

Azure Blockchain provides a rapid, low-cost, low-risk platform for building and deploying blockchain applications.

Enough with the sales pitch, let's investigate some key capabilities of Azure BaaS, before diving into the configuration of a specific ledger in the Azure portal. Azure Blockchain consists of:

- Single-node ledgers to simulate production for multiple divisions within a single organization.
- Multi-node ledgers to simulate production for multiple divisions within multiple organizations.
- Tools for development of decentralized applications distributed on a blockchain.

Decentralized applications (dApps) are applications that run on a peer-to-peer network of computers rather than a single computer. In blockchain context, think of a dApp as a client application that communicates to a smart contract for interacting with the blockchain network. A good introduction to dApps can be found on BlockchainHub at bit.ly/2rRkijj.

The key characteristics of building a blockchain infrastructure in Azure are:

- Establish a secure environment that exposes protected endpoints. This can be done via Azure Virtual Networks, Azure App Services VNet Integration or Network Security Groups.
- Develop smart contracts, using any of the available development tools, such as Blockstack Core, Ethereum Studio or Truffle.
- Automate deployment of participant components, both virtual machines and Platform-as-a-Service components. This can be enabled by Azure Resource Manager and PowerShell scripts.

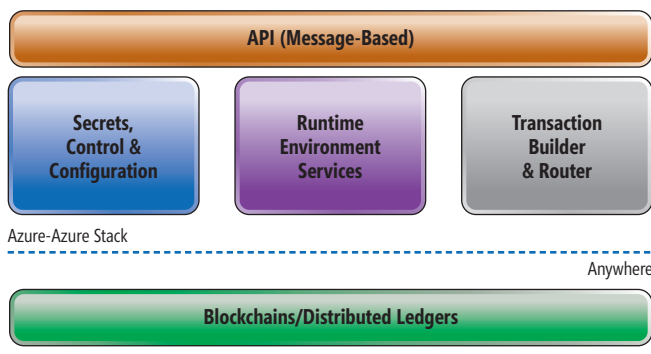


Figure 3 Azure Blockchain-as-a-Service Stack

- Protect access to data and logic, with user-level authentication and authorization, by implementing Azure AD to secure apps and APIs.
- In general, build an architecture for enterprise solution integration with a blockchain ledger, leveraging Azure enterprise capabilities and worldwide distribution.

Azure BaaS, in a nutshell, represents not just a public cloud hosting provider for distributed ledgers, but an organic and integrated platform for building and delivering decentralized applications that run on a blockchain technology. **Figure 3** illustrates the platform architecture.

Now let's explore the digital ledger provisioning capabilities of Azure BaaS. First, I need to access Azure portal (portal.azure.com) and create a new service from the Azure Marketplace | Blockchain section. I can select from several digital ledger technologies—in this case I'll create an Ethereum Consortium multi-node ledger—and quickly provision my blockchain network in Azure.

Ethereum Consortium

The Ethereum Consortium template deploys an Ethereum multi-member network, consisting of a set of mining nodes and transaction nodes. Provisioning can take up to 20 minutes, depending on the size of the network, at which point I can configure additional Ethereum accounts and get started with smart contract and dApp development through the administrator Web page.

The provisioning process guides you through five steps to enter the necessary configuration settings for provisioning the Ethereum ledger in Azure:

Step 1: Configure basic settings, which include a Resource prefix for naming all the generated Azure resources provisioned in the assigned Resource group, authentication credentials as admin of all the deployed virtual machines and the Azure region of deployment.

The Ethereum Consortium template in Azure Blockchain consists of a set of mining nodes and transaction nodes that can be deployed in minutes.

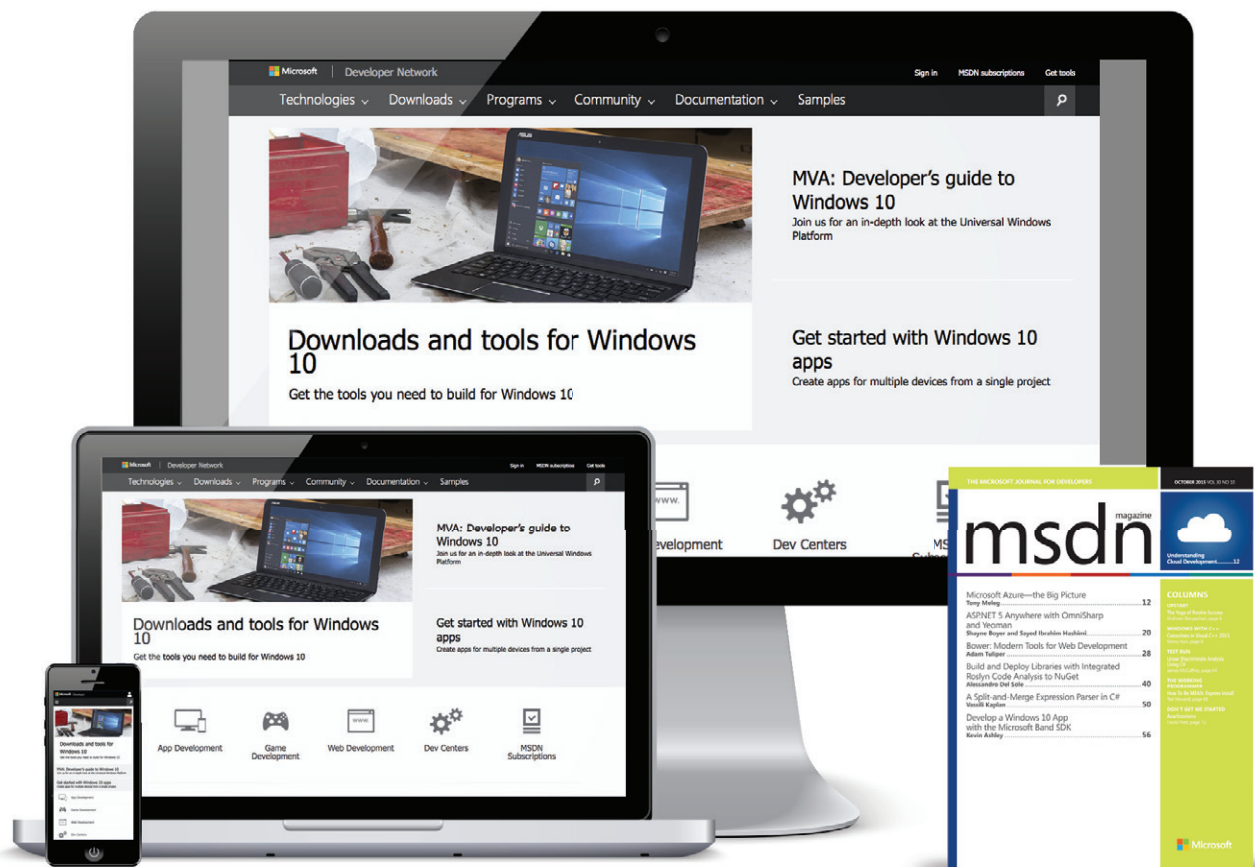
Step 2: Address network size and performance. You may want to specify the number of members in the consortium (up to 12), the number of mining nodes per member (mining nodes record transactions within a blockchain network), storage replication (locally redundant or geo-redundant) and performance. You can also set the number of load-balanced transaction nodes, which represents the point of interaction of users or applications with the deployed blockchain.

Step 3: This step is specific to Ethereum nodes. I can specify the Ethereum Network ID, a unique value that identifies the network and will be used by nodes to peer with each other. Also, I can specify how the first block, called Genesis, will be generated,

msdn

magazine

Where you need us most.



Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!
360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com



Figure 4 Resource Template File for Ethereum Consortium Multi-Node Ledger

either automatically by the platform or manually by providing my own JSON file.

Step 4: Before I deploy the resources in the Ethereum Consortium ledger, I'm presented with a summary of the configuration settings I entered. From here, I can download a JSON template file to automate deployment of a similar resource set with Azure Resource Manager. **Figure 4** anticipates the Azure resources to be deployed in the provisioning of the Ethereum ledger, along with a snippet of the template file.

This template file can be used to automate deployment of similar resources in the future, using a combination of .NET or PowerShell scripts.

Figure 5 The DeploymentHelper Class

```
class DeploymentHelper
{
    string subscriptionId = "your-subscription-id";
    string clientId = "your-service-principal-clientId";
    string clientSecret = "your-service-principal-client-secret";
    string resourceGroupName = "resource-group-name";
    string deploymentName = "deployment-name";
    string resourceGroupLocation = "resource-group-location";
    // Must be specified for creating a new resource group
    string pathToTemplateFile = "path-to-template.json-on-disk";
    string pathToParameterFile = "path-to-parameters.json-on-disk";
    string tenantId = "tenant-id";

    public async void Run()
    {
        // Try to obtain the service credentials
        var serviceCreds = await ApplicationTokenProvider.LoginSilentAsync(
            tenantId, clientId, clientSecret);

        // Read the template and parameter file contents
        JObject templateFileContents =
            GetJsonFileContents(pathToTemplateFile);
        JObject parameterFileContents =
            GetJsonFileContents(pathToParameterFile);

        // Create the resource manager client
        var resourceManagementClient =
            new ResourceManagementClient(serviceCreds);
        resourceManagementClient.SubscriptionId = subscriptionId;

        // Create or check that resource group exists
        EnsureResourceGroupExists(resourceManagementClient, resourceGroupName,
            resourceGroupLocation);

        // Start a deployment
        DeployTemplate(resourceManagementClient, resourceGroupName, deploymentName,
            templateFileContents, parameterFileContents);
    }
}
```

The C# code in **Figure 5** describes the DeploymentHelper class generated by the template for automating the deployment of the identified Azure resources. You need to reference the following packages to run the code:

- Microsoft.Azure.Management.Authorization
- Microsoft.Azure.Management.ResourceManager
- Microsoft.Rest.ClientRuntime.Azure.Authentication

Similarly, the PowerShell script signs into an Azure subscription, registers the necessary resource providers and then starts the deployment of the resources identified in the template file, as shown in **Figure 6**.

The entire solution, consisting of template and script files, is available for download from my GitHub repository at bit.ly/2lNgNEP.

Once that's done, review the terms of use and licensing conditions, and click Create to deploy the resources. In less than 20 minutes, you have a fully functional blockchain ledger up and running. Just don't forget to save important information needed for developing dApps, including:

- **RPC-Endpoint:** You need this address to establish communication between a dApp development environment, such as Ethereum Remix and the consortium blockchain.
- **SSH Info:** You need credentials to sign into the blockchain environments and configure parameters, like most typically for unlocking the Coinbase account and start mining new blocks.

Coinbase is my digital wallet, which contains my signature keys used to hash a block, and my Ether, the cryptocurrency of Ethereum, earned as part of the mining process. When deploying a new Ethereum Consortium ledger in Azure, this account is initially locked, so I need to unlock it before I can publish smart contracts. With the help of SSH, I connect to a transaction node of the Ethereum Consortium network and unlock the Coinbase account, like so:

```
geth attach -- opens the Geth console
personal.unlockAccount(eth.coinbase)
```

When prompted for a passphrase, I enter the gethadmin password that I specified in Step 1 of the configuration wizard (not the Ethereum private key passphrase). By default, this action unlocks the Coinbase account for 5 minutes. You can change the duration using a different signature of the unlockAccount method, as shown here:

File Format APIs

Open, Create, Convert, Print and Save files from your applications!

Try risk free – 30 day trial



Download a Free Trial at



<https://downloads.aspose.com>



Aspose.Words

Create, edit, convert or print Word documents (DOC, DOCX, RTF etc.) in your .NET, Java and Android applications.



Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).



Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS etc.) using our native APIs for .NET, Java and Android platforms.



Aspose.Slides

Create, edit or convert PowerPoint presentations (PPT, PPTX, ODP etc.) in your .NET, Java and Android applications.



Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.



Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet etc.) using our native APIs for .NET and Java.



Aspose.Imaging

Deliver efficient applications to Create, Draw, Manipulate or Convert image file formats.



Aspose.Tasks

Develop high performance apps to Create, Edit or Convert Microsoft Project® document formats.

► Aspose.Diagram ► Aspose.Note ► Aspose.3D ► Aspose.CAD ► Aspose.HTML ► Aspose.GIS

Americas: +1 903 306 1676

EMEA: +44 141 628 8900
sales@asposeptyltd.com

Oceania: +61 2 8006 6987


```
eth.coinbase -- address of the coinbase account
personal.unlockAccount('address', 'passphrase', 'duration') --
unlocking the account for a longer time period
```

If you're wondering what the “geth” command stands for, it's a multipurpose command-line tool that runs a full Ethereum node implemented in Go.

Once the Coinbase account is unlocked, this represents *time zero*, when the network starts. After this point, nodes can accept transactions. Transactions could be in the form of creation of accounts, movement of ether, creation of smart contracts, or any change to the state of the blockchain. Then, at a periodic time configured for the network, the network mines the next block. This block is a hash calculated by combining hashes of the transactions executed between the last block and now, plus the hash from the previous block and a nonce—a sequence of bits in a block that can be adjusted in order to try to satisfy the proof-of-work condition.

This is the essence of mining. This value makes satisfying “proof of work” a difficult computational task that depends on luck or brute force. The block is then accepted by the network by consensus, and so you have the first two blocks in your chain, and so on.

Figure 6: Starting the Deployment

```
# sign in
Write-Host "Logging in...";
Login-AzureRmAccount;

# select subscription
Write-Host "Selecting subscription '$subscriptionId'";
Select-AzureRmSubscription -SubscriptionID $subscriptionId;

# Register RPs
$resourceProviders = @("microsoft.compute","microsoft.
resources","microsoft.network");
if($resourceProviders.length) {
    Write-Host "Registering resource providers"
    foreach($resourceProvider in $resourceProviders) {
        RegisterRP($resourceProvider);
    }
}

#Create or check for existing resource group
$resourceGroup = Get-AzureRmResourceGroup -Name $resourceGroupName
-ErrorAction SilentlyContinue
if(!$resourceGroup)
{
    Write-Host "Resource group '$resourceGroupName' does not exist.
    To create a new resource
    group, please enter a location.";
    if(!$resourceGroupLocation) {
        $resourceGroupLocation = Read-Host "resourceGroupLocation";
    }
    Write-Host "Creating resource group '$resourceGroupName' in location
    '$resourceGroupLocation'";
    New-AzureRmResourceGroup -Name $resourceGroupName
    -Location $resourceGroupLocation
}
else{
    Write-Host "Using existing resource group '$resourceGroupName'";
}

# Start the deployment
Write-Host "Starting deployment...";
if(Test-Path $parametersFilePath) {
    New-AzureRmResourceGroupDeployment -ResourceGroupName $resourceGroupName
    -TemplateFile $templateFilePath -TemplateParameterFile
    $parametersFilePath;
} else {
    New-AzureRmResourceGroupDeployment -ResourceGroupName $resourceGroupName
    -TemplateFile $templateFilePath;
}
```

Developing Smart Contracts

It would take a whole book to go through the details of developing smart contracts in Ethereum. In this article, I want to offer pointers to start with and understand the landscape of technologies and frameworks in use.

To write and deploy smart contracts in Ethereum, you can use any of the development environment available in Azure, or access a completely external browser-based IDE like Ether Camp (ether.camp) or Ethereum Remix (remix.ethereum.org).

In terms of programming languages, Solidity (solidity.readthedocs.io) is a popular contract-oriented language for blockchain programming, with a JavaScript-like syntax.

On the client side, programming languages that support interacting with an Ethereum node include C#, C++, JavaScript and more. It's possible to write C# code using a library like Nethereum (nethereum.com), a fully managed .NET integration library for Ethereum that allows interaction with Ethereum clients like geth, eth or parity using RPC. The library has very similar functionality to the JavaScript Ethereum Web3 RPC Client Library, which is the de-facto standard for blockchain client interoperability.

For example, to make a Smart Contract call via Nethereum, I need to do the following:

- Obtain the smart contract address and Application Binary Interface (ABI). An ABI is the interface to call functions in a smart contract and get data back from an Ethereum node.
- Obtain the function signature on the smart contract to be invoked.
- Unlock the Ethereum account making the call with the account's passphrase.
- Make the call to the smart contract.

The code snippet in **Figure 7** shows a few very basic steps, using the Nethereum library.

The invoked smart contract, written in Solidity, would look like a ballot contract that exposes a vote method, which accepts a proposal number in input. When a vote is cast, the voted flag on the voter (the message sender) is set to true to prevent double voting,

Figure 7 Calling a Smart Contract

```
// Obtain the contract ABI
abi = db.GetContract(ballot.ContractID);

// Get the function address to call on the smart contract
var func = web3.Eth.GetContract(
    abi,
    ballot.ContractID).GetFunction("vote");

// Unlock the account so you can call the smart contract
string passphrase = db.GetAccountPassphrase(agreement.OriginatorAccount);
bool success = await web3.Personal.UnlockAccount.SendRequestAsync(
    ballot.OriginatorAccount,
    passphrase,
    120);

// Make the smart contract call
if (success)
{
    object[] args = new object[] {
        id,
        ballot.OriginatorAccount,
        ballot.CounterSigAccount,
        123 /* sample proposal number to vote for */ };

    // Call the "vote" function on the smart contract
    await func.SendTransactionAsync(ballot.OriginatorAccount, args);
}
```

Figure 8 A Ballot Contract in Solidity Code

```
pragma solidity ^0.4.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    /// Create a new ballot with different proposals
    function Ballot(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }

    /// Give a single vote to the given proposal
    function vote(uint8 toProposal) public {
        Voter storage sender = voters[msg.sender];
        if (sender.voted || toProposal >= proposals.length) return;
        sender.voted = true;
        sender.vote = toProposal;
        proposals[toProposal].voteCount += sender.weight;
    }
}
```

and the proposal counter is increased, considering a weight for the vote itself. **Figure 8** shows the code for this.

Please note that this is Solidity code, a strongly typed language with JavaScript-like syntax, with a few variants, like the contract, struct and address keywords, or the triple slash (“///”) for a comment. I use Ethereum Remix for development of smart contracts in Solidity, which provides a Web-based IDE for development, testing and deployment.

Azure Blockchain vNext

Let's look at what the future looks like in the Microsoft vision for blockchain technology. Coming soon, the Microsoft Confidential Consortium (Coco) Framework is an open source system that enables high-scale, confidential blockchain networks that meet all key enterprise requirements for confidentiality, governance and performance, and at the same time provide a means to accelerate production enterprise adoption of blockchain technology.

Coco (bit.ly/2lor8YA) brings together the power of existing blockchain protocols, trusted execution environments, distributed systems and cryptography to enable enterprise-ready blockchain networks that deliver:

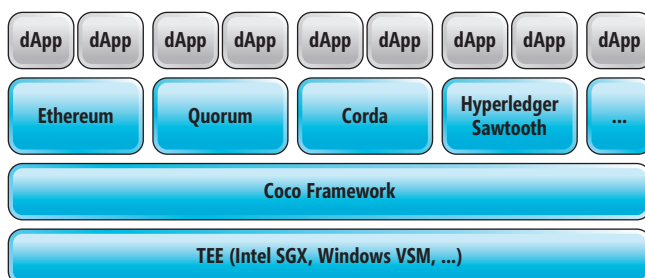


Figure 9 High-Level Overview of the Coco Framework

- Throughput and latency approaching database speeds
- Richer, more flexible, business-specific confidentiality models
- Network policy management through distributed governance
- Support for non-deterministic transactions
- Reduced energy consumption

It's important to note that Coco isn't a standalone blockchain protocol. Rather, it provides a trusted foundation with which existing blockchain protocols such as Ethereum, Quorum, Corda and others can be integrated to deliver complete, enterprise-ready ledger solutions. Coco is designed to be open and compatible with any blockchain protocol. It achieves this through the use of trusted execution environments (TEE), such as Intel Software Guard Extensions (SGX) and Windows Virtual Secure Mode (VSM), to enable the creation of a trusted network of physical nodes on which to run a distributed ledger. **Figure 9** shows the Coco Framework architecture.

Coco provides code assets and ARM template deployment scripts for the scaffolding needed to create a blockchain network, gateway API and Web application. It also provides for Azure AD and Azure Key Vault integration, and supports SQL databases for collecting on- and off-chain data. Finally, it provides supporting code and services for block hashing and signing. Coco uses Azure Event Hubs at its core to add new capabilities, such as sending raw data to Azure Data Lake or providing transaction data to Azure Search.

Coco makes it possible to create blockchain applications without writing any code. It uses the metadata provided for smart contracts to dynamically deliver a contextual UX for participants. Because the framework populates SQL databases as an off-chain store, it enables an organization to leverage existing skills and tools to light up additional capabilities such as APIs, PowerBI reporting, chat bots, Azure Data Factory and machine learning.

Microsoft plans to open source the Coco Framework code later in 2018.

Finally, a word on Azure Blockchain Workbench, which is the primary mechanism for enterprise customers getting started with blockchain. Azure Blockchain is a collection of Azure services and capabilities designed to help enterprises create and deploy a new class of applications for sharing business processes and data with multiple, semi-trusted organizations. Currently, customers can deploy these services into their subscriptions and integrate them with blockchains available on the Azure Marketplace. With Azure Blockchain Workbench the heavy lifting is done for them, so they can focus less on scaffolding and more on logic and smart contracts.

Azure Blockchain Workbench is available now in the Azure Marketplace (aka.ms/tryworkbench). ■

STEFANO TEMPESTA is a Microsoft Regional Director and MVP, as well as chapter leader for CRMUG in Switzerland, the largest community of Dynamics 365/CRM experts in the world. Tempesta is an instructor of courses about Dynamics 365, blockchain and machine learning, and a regular speaker at international IT conferences, including Microsoft Ignite and Tech Summit. He founded Blogchain Space (blogchain.space), a blog about blockchain technologies, writes for MSDN Magazine and MS Dynamics World, and publishes machine learning experiments on the Azure AI Gallery (gallery.azure.ai).

THANKS to the following Microsoft technical experts for reviewing this article: James McCaffrey

AUGUST 6 – 10, 2018 • Microsoft Headquarters, Redmond, WA



Change is ~~Coming~~. Are You Ready? **HERE**

Join us for TechMentor, August 6 – 8, 2018, as we return to Microsoft Headquarters in Redmond, WA. In today's IT world, more things change than stay the same. As we celebrate the 20th year of TechMentor, we are more committed than ever to providing immediately usable IT education, with the tools you need today, while preparing you for tomorrow – **keep up, stay ahead and avoid Winter, ahem, Change.**

Plus you'll be at the source, Microsoft HQ, where you can have lunch with Blue Badges, visit the company store, and experience life on campus for a week!

You owe it to yourself, your company and your career to be at TechMentor Redmond 2018!



**REGISTER NOW FOR
BEST SAVINGS!**

Use Promo Code MSDN

EVENT PARTNER



SUPPORTED BY



PRODUCED BY



AGENDA AT-A-GLANCE

Client and Endpoint Management		PowerShell and DevOps	Infrastructure	Soft Skills for ITPros	Security	Cloud (Public/ Hybrid/Private)
START TIME	END TIME	TechMentor Pre-Conference Workshops: Monday, August 6, 2018 <i>(Separate entry fee required)</i>				
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Light Breakfast				
9:00 AM	12:00 PM	M01 Workshop: How to Prevent all Ransomware / Malware in 2018 - Sami Laiho	M02 Workshop: Building Office 365 Federated Identity from Scratch Using AD FS - Nestori Syynimaa		M03 Workshop: Managing Windows Server with Project Honolulu - Dave Kawula	
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:00 PM	5:00 PM	M01 Workshop: How to Prevent all Ransomware / Malware in 2018 (Continues) - Sami Laiho	M04 Workshop: Master PowerShell Tricks for Windows Server 2016 and Windows 10 - Will Anderson & Thomas Rayner		M05 Workshop: Dave Kawula's Notes from the Field on Microsoft Storage Spaces Direct - Dave Kawula	
6:30 PM	8:30 PM	Dine-A-Round Dinner - Suite in Hyatt Regency Lobby				
START TIME	END TIME	TechMentor Day 1: Tuesday, August 7, 2018				
7:00 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	T01 Enterprise Client Management in a Modern World - Kent Agerlund	T02 How to Write (PowerShell) Code that Doesn't Suck - Thomas Rayner	T03 The Easy Peasy of Troubleshooting Azure - Mike Nelson	T04 Nine O365 Security Issues Microsoft Probably Hasn't Told You (and You Probably Don't Want to Know) - Nestori Syynimaa	
9:30 AM	10:45 AM	T05 Managing Client Health—Getting Close to the Famous 100% - Kent Agerlund	T06 The Network is Slow! Or is it? Network Troubleshooting for Windows Administrators - Richard Hicks	T07 Getting Started with PowerShell 6.0 for IT Pro's - Sven van Rijen	T08 The Weakest Link of Office 365 Security - Nestori Syynimaa	
11:00 AM	12:00 PM	KEYNOTE: What's Next for OneDrive and Microsoft 365 - Stephen L. Rose, Sr. PMM, One Drive For Business, Microsoft				
12:00 PM	1:00 PM	Lunch - McKinley / Visit Exhibitors - Foyer				
1:00 PM	2:15 PM	T09 How to Get Started with Microsoft EMS Right Now - Peter Daalmans	T10 Back to the Future! Access Anywhere with Windows 10 Always on VPN - Richard Hicks	T11 Using Desired State Configuration in Azure - Will Anderson	T12 To Be Announced	
2:15 PM	2:45 PM	Sponsored Break - Visit Exhibitors - Foyer				
2:45 PM	4:00 PM	T13 Conceptualizing Azure Resource Manager Templates - Will Anderson	T14 How to Use PowerShell to Become a Windows Management SuperHero - Petri Paavola	T15 Making the Most Out of the Azure Dev/Test Labs - Mike Nelson	T16 To Be Announced	
4:00 PM	5:30 PM	Exhibitor Reception - Attend Exhibitor Demo - Foyer				
START TIME	END TIME	TechMentor Day 2: Wednesday, August 8, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	W01 Automated Troubleshooting Techniques in Enterprise Domains (Part 1) - Petri Paavola	W02 Troubleshooting Sysinternals Tools 2018 Edition - Sami Laiho	W03 In-Depth Introduction to Docker - Neil Peterson	W04 How Microsoft Cloud Can Support Your GDPR Journey - Milad Aslaner	
9:30 AM	10:45 AM	W05 Automated Troubleshooting Techniques in Enterprise Domains (Part 2) - Petri Paavola	W06 What's New in Windows Server 1803 - Dave Kawula	W07 Simplify and Streamline Office 365 Deployments the Easy Way - John O'Neill, Sr.	W08 How to Administer Microsoft Teams Like a Boss - Ståle Hansen	
11:00 AM	12:00 PM	TECHMENTOR PANEL: The Future of Windows - Peter De Tender, Dave Kawula, Sami Laiho, & Petri Paavola				
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - McKinley / Visit Exhibitors - Foyer				
1:00 PM	1:30 PM	Networking Break - Exhibitor Raffle @ 1:10 pm (Must be present to win) - Foyer in front of Business Center				
1:30 PM	2:45 PM	W09 Putting the Windows Assessment and Deployment Kit to Work - John O'Neill, Sr.	W10 Deploying Application Whitelisting on Windows Pro or Enterprise - Sami Laiho	W11 Azure is 100% High-Available... Or Is It? - Peter De Tender	W12 What the NinjaCat Learned from Fighting Cybercrime - Milad Aslaner	
3:00 PM	4:15 PM	W13 The Evolution of a Geek—Becoming an IT Architect - Mike Nelson	W14 Advanced DNS, DHCP and IPAM Administration on Windows Server 2016 - Orin Thomas	W15 Managing Tesla Vehicles from the Cloud - Marcel Zehner	W16 Nano Server—Containers in the Cloud - David O'Brien	
6:15 PM	9:00 PM	Set Sail! TechMentor's Seattle Sunset Cruise - Buses depart the Hyatt Regency at 6:15pm to travel to Kirkland City Dock				
START TIME	END TIME	TechMentor Day 3: Thursday, August 9, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	TH01 Manage Your Apple Investments with Microsoft EMS - Peter Daalmans	TH02 Tips and Tricks for Managing and Running Ubuntu/Bash/Windows Subsystem for Linux - Orin Thomas	TH03 The OMS Solutions Bakery - Marcel Zehner	TH04 Getting Started with PowerShell for Office 365 - Vlad Catrinescu	
9:30 AM	10:45 AM	TH05 HoloLens, Augmented Reality, and IT - John O'Neill, Sr.	TH06 A Real-world Social Engineering Attack and Response - Milad Aslaner	TH07 30 Terrible Habits of Server and Cloud Administrators - Orin Thomas	TH08 Advanced PowerShell for Office 365 - Vlad Catrinescu	
11:00 AM	12:15 PM	TH09 10 Tips to Control Access to Corporate Resources with Enterprise Mobility + Security - Peter Daalmans	TH10 What's New and Trending with Microsoft Enterprise Client Management - Kent Agerlund	TH11 OneNote LifeHack: 5 Steps for Succeeding with Personal Productivity - Ståle Hansen	TH12 Managing Virtual Machines on AWS—Like in Real Life! - David O'Brien	
12:15 PM	2:15 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:15 PM	3:30 PM	TH13 Security Implications of Virtualizing Active Directory Domain Controllers - Sander Berkouwer	TH14 Building a New Career in 5 Hours a Week - Michael Bender	TH15 Azure CLI 2.0 Deep Dive - Neil Peterson	TH16 OpenSSH for Windows Pros - Anthony Nocentino	
3:45 PM	5:00 PM	TH17 Running Hyper-V in Production for 10 years - Notes from the Field - Dave Kawula	TH18 Network Sustainability and Cyber Security Measures - Omar Valerio	TH19 Azure AD Connect Inside and Out - Sander Berkouwer	TH20 I Needed to Install 80 SQL Servers...Fast. Here's How I Did It! - Anthony Nocentino	
START TIME	END TIME	TechMentor Post-Conference Workshops: Friday, August 10, 2018 <i>(Separate entry fee required)</i>				
8:30 AM	9:00 AM	Post-Conference Workshop Registration - Coffee and Light Breakfast				
9:00 AM	12:00 PM	F01 Workshop: Hardening Your Windows Server Environment - Orin Thomas		F02 Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 - Peter De Tender		
12:00 PM	1:00 PM	Lunch - McKinley				
1:00 PM	4:00 PM	F01 Workshop: Hardening Your Windows Server Environment (Continues) - Orin Thomas		F02 Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 (Continues) - Peter De Tender		

Speakers and sessions subject to change

CONNECT WITH TECHMENTOR



@TechMentorEvent

Facebook
Search "TechMentor"LinkedIn
Search "TechMentor"

techmentorevents.com/redmond



Online Users, Streaming and Other SignalR Goodies

If you ever used any version of SignalR for the classic ASP.NET platform, you should be quite familiar with the concept of a hub. In SignalR a hub is the component that enables compatible client and server applications to arrange bidirectional remote procedure calls, from client to server and from server back to connected clients.

Concretely, in terms of software, a hub is a class that inherits from a system-provided base class and exposes endpoints for clients to call. Conceptually speaking, it has a few points in common with an ASP.NET controller. In particular, it's the façade that receives client calls and reacts to them, and is organized around a relatively small number of related functions. In ASP.NET Core SignalR, the similarity between hubs and controllers is, in a way, even closer. This is my third article about ASP.NET Core SignalR for the Cutting Edge column, and in neither of the previous two articles did I use a non-empty hub class. Explaining a bit more about SignalR hubs and how they're implemented in ASP.NET Core is one of the purposes of this article. However, I'll also touch on other interesting aspects of SignalR, specifically data streaming and the count of online users.

Hubs on ASP.NET Core SignalR

A hub is the entry point in a message pipeline through which connected clients and servers exchange messages. Hubs expose their methods as URLs and process any received parameters via model binding, in much the same way a canonical controller does. In a way, a hub is a dedicated controller that operates over two built-in protocols. The default protocol consists of a JSON object, but another binary protocol can be used that's based on MessagePack. Note that in order to support MessagePack, browsers must support XHR Level 2. As Level 2 was introduced back in 2012, this might not be much of an issue today, but if your application requires support for very old browsers it might be worth noting. A quick browser check can be made here: caniuse.com/#feat=xhr2.

If any of the connected clients request a SignalR endpoint, the hub is directly invoked by the SignalR runtime engine. The conversation takes place over the selected protocol, mostly likely WebSockets. In order to invoke the server, clients need to hold a connection object. A Web client would get it as follows:

```
var clockConnection = new signalR.HubConnection("/clockDemo");
clockConnection.start().then(() => {
    clockConnection.invoke("now");
});
```

A client invokes a server endpoint via the "invoke" method on the connection object. Note that the exact syntax may vary depending

on the actual technology used for the client. The server replies through the methods defined on the base Hub class, and the conversation takes place over the transport protocol of choice, most often WebSockets, like this:

```
public class ClockHub : Hub
{
    public Task Now()
    {
        var now = DateTime.UtcNow.ToShortTimeString();
        return Clients.All.SendAsync("now", now);
    }
}
```

Note that you won't be able to monitor the various calls using an HTTP tool like Fiddler. You need something like Chrome Developer Tools. In all the examples I wrote for my past columns on SignalR, however, I always used an empty hub class.

The hub class is the official SignalR façade for receiving client calls, and the fastest way for client/server communication to take place because of the dedicated pipeline. When the call occurs through the hub, the SignalR runtime can track and expose all the available information via the properties of the base Hub class. In this way, the SignalR connection ID and the entire context of the call, including any claims of the authenticated user, are available.

A hub is the entry point in a message pipeline through which connected clients and servers exchange messages.

In addition, through the hub, developers can handle connect and disconnect events. Any time a new connection is set up, or ceased, a hub method is called back. If you use SignalR only as a way to monitor remote long-running operations, then you can also trigger the task via a plain controller and inject a hub context in it for notifications. Or as an alternative, you can invoke the hub and trigger the task from within the hub. It's your choice. SignalR works as an end-to-end framework, a bridge between the client and the server. Coding logic into the hub is acceptable as long as the work doesn't go too deep into the layers of your code. Otherwise, go through the controller—whether MVC or Web API—and inject a hub context.

Figure 1 Counting Connections

```
public class SampleHub : Hub
{
    private static int Count = 0;
    public override Task OnConnectedAsync()
    {
        Count++;
        base.OnConnectedAsync();
        Clients.All.SendAsync("updateCount", Count);
        return Task.CompletedTask;
    }

    public override Task OnDisconnectedAsync(Exception exception)
    {
        Count--;
        base.OnDisconnectedAsync(exception);
        Clients.All.SendAsync("updateCount", Count);
        return Task.CompletedTask;
    }
}
```

The only difference between using a hub or a controller is that SignalR can't track the connection ID when a request goes through the controller. If the connection ID is relevant to the server task, then it has to be passed in some way via the URL. All other information that forms the SignalR caller context can be retrieved by the controller via the HTTP request context.

Counting Online Users

Some Web applications find it useful, or just engaging for users, to show how many connections are currently active. The problem is not so much tracking when a user connects—there are many endpoints through which you can detect that—but rather when a user disconnects from the application.

You can audit a login page or the post-authentication step. You can place a check in some base controller class or in any of the pages that the user can visit. In general, you can always find a way to detect when a user connects to the application. The problem is how the user can leave the application, whether by logging out (easily traceable) or by navigating away from the page or by shutting down the browser window. There's no reliable way to detect when the user closes the browser window. Yes, browsers usually fire the `beforeunload` event when the browser shuts down, but this same event is also fired whenever you follow a link—even when that link is within the same application. So it's not a perfect solution.

A much more reliable way to count users is to keep track of ASP.NET Core SignalR connections. To do this, you need a fully

functional hub with the connection set up through it. When the user leaves the browser, or just the application, the connection is released and listening clients notified. As in ASP.NET Core SignalR, there's no support for automatic reconnections, so things are even easier. All you do is define a global static variable in the hub and increment its value up or down when a user connects or disconnects, as shown in **Figure 1**. The SignalR runtime in ASP.NET Core ensures that every connection is closed at some point, and any new connection effectively refers to a new connection. In short, the number you get is highly reliable.

There's one drawback to counting users with SignalR: It only works if users visit the page that establishes a connection to the hub where counting takes place. To be on the safe side, you need to have a SignalR client in nearly any page the user can visit. This is especially true if you consider that normally the number of online users is a globally visible value you probably have in all layouts on which your views are based.

Note that in the sample hub code, the class calls back the connected clients every time a connection is created or closed. Note also that in this way you only have the total number of users, but not the list of connection IDs or, in case of authenticated users, the list of user names. To achieve this, you better use a dictionary instead of a global counter and add to it entries with connection IDs or claims, such as the user name.

There's one drawback
to counting users with SignalR:
It only works if users visit the
page that establishes a
connection to the hub where
counting takes place.

Another point to consider with reference to the code in **Figure 1** is the use of a static variable to count users. A static variable is per-server, which means that when you scale out you'll need to consider how to store shared state in a globally accessible location, such as a database or a distributed cache.

Figure 2 Methods for the Server to Call Back Connected Clients

Expression	Description
Clients.All	The notification is broadcast to all connected clients, regardless of the technology being used (Web, .NET, .NET Core, Xamarin).
Clients.Client(connectionId)	The notification is sent exclusively to the client listening over the specified connection.
Clients.User(userId)	The notification is sent to all clients whose authenticated user matches the provided user name.
Clients.Groups(group)	The notification is sent to all clients belonging to the specified group.

Pushing Information Back

From within the hub, or the hub context if you connect to the back end via a controller method, you have many different ways to call back connected clients. All methods are members exposed by the `Clients` object that, in spite of the name, is not a collection, but an instance of the `IClientProxy` class. The expressions in **Figure 2** indicate the object from which the `SendAsync` method is invoked. The `SendAsync` method takes the name of the client method to call back and the parameters to pass.

A group is a collection of related clients collectively gathered under a name. The more natural way of thinking of groups in

Figure 3 The Hub Class That Streams Back

```
public class ClockHub : Hub
{
    private static bool _clockRunning = false;

    public void Start()
    {
        _clockRunning = true;
        Clients.All.SendAsync("clockStarted");
    }

    public void Stop()
    {
        _clockRunning = false;
        Clients.All.SendAsync("clockStopped");
    }

    public ChannelReader<string> Tick()
    {
        var channel = Channel.CreateUnbounded<string>();
        Task.Run(async () => {
            while(_clockRunning)
            {
                var time = DateTime.UtcNow.ToString("HH:mm:ss");
                await channel.Writer.WriteAsync(time);
                await Task.Delay(1000);
            }
            channel.Writer.TryComplete();
        });
    }
}
```

SignalR is chat rooms. A group is created programmatically simply adding connection IDs to the group of a given name. Here's how:

```
hub.Groups.AddAsync(connectionId, nameOfGroup);
```

Connected clients receive their data through a callback. This is only the most common technique. In ASP.NET Core SignalR, you can also use streaming.

Data Streaming

Probably the most interesting new aspect of SignalR is support for streaming. Streaming is similar to broadcasting, but it follows a slightly different model and is essentially a slightly different way of achieving the same broadcast-style communication. With SignalR streaming, the hub still needs to poll or listen for data in order to stream it back. In classic broadcast, the server tells a client method when new data is available.

The hub offers three methods to start, stop and operate the clock.

In the new streaming model, the client subscribes to a new server object of type `Channel` and the server—the hub, actually—yields new items as they're captured. At the moment, there's nothing like a true stream that flows bytes toward all the connected clients as they become available, but this model can be supported in the future. Note that the `Channel` type has been introduced with preview2 and is not supported in earlier builds. In earlier builds, you must use observables instead, which require a reference to the `System.Reactive.Linq` NuGet package. The switch between observables and the new type `Channel` relates to the lack of primitives in `IObservable` for working with network backpressure (that

is, telling the server to slow down when the client isn't processing messages fast enough).

Figure 3 presents the code for the hub.

The hub offers three methods to start, stop and operate the clock. A global variable controls the running status of the streaming, and start and stop methods set the control variable and notify back client methods as usual in a SignalR hub. The tricky part is the `Tick` method. The method name coincides with the name of the stream to which clients will subscribe. The method returns a channel object of a given type. In the example, the type is a simple string but it can be anything more sophisticated.

Every invocation, from client to server or server to client, consists of one party sending an invocation message, and the other party eventually responding with a completion message that carries a result or an error. In a SignalR streaming scenario, instead, the other party responds with multiple messages, each carrying a data item, before eventually concluding the communication with a completion message. As a result, the client ends up processing multiple items even before the completion message is received.

Scaling to Multiple Instances

SignalR keeps all connection IDs in memory, meaning that the moment the application scales to multiple instances, the broadcast (but also streaming, as discussed later) is compromised as each instance would only track a portion of all connected clients. To avoid that, SignalR supports a Redis-based cache that ensures that new connections are automatically shared between instances. To enable Redis, you need the `SignalR.Redis` package and a call to the `AddRedis` method in the `ConfigureServices` method of the startup class, like so:

```
services.AddSignalR()
    .AddRedis("connection string");
```

The option parameter serves the purpose of specifying the connection string to the running instance of Redis.

Wrapping Up

ASP.NET Core SignalR comes with two significant changes from the non-Core version. One is the lack of automatic reconnection, which has an impact on how connect/disconnect and online user count is handled programmatically. This means that now every application has to handle connection/disconnection logic and likely has to identify the difference between a user connecting for the first time and a user reconnecting due to an error. The other change is support for data streaming. Data streaming is based on channels and at the moment only supports specific data items instead of raw streams.

Finally, my exploration of SignalR lacks one more piece, which I'll address in a future column: authenticated users and groups. ■

DINO ESPOSITO has authored more than 20 books and 1,000 articles in his 25-year career. Author of "The Sabbatical Break," a theatrical-style show, Esposito is busy writing software for a greener world as the digital strategist at BaxEnergy. Follow him on Twitter: @despos.

THANKS to the following Microsoft expert for reviewing this article:
Andrew Stanton-Nurse

Imaging SDK for Winforms, WPF, and Web Development

GdPicture.NET



- ✦ Scanning
- ✦ OCR
- ✦ 100+ Formats
- ✦ Image Cleanup
- ✦ Annotations
- ✦ Barcodes
- ✦ Document Compression
- ✦ MICR
- ✦ Form Processing
- ✦ Thumbnails
- ✦ Viewer Control
- ✦ PDF
- ✦ Bookmarks
- ✦ Image Processing
- ✦ Color Detection
- ✦ Printing
- ✦ DICOM
- ✦ TIFF
- ✦ DOCX
- ✦ Metadata Support
- ✦ Document Conversion

Leverage your apps. with GdPicture.NET Imaging Toolkit

**60-day Free Trial
Support Included**

www.gdpicture.com



Visual Studio[®] LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

September 17-20, 2018
Renaissance

Chicago

Visual Studio LIVE
25
YEARS OF CODING INNOVATION
1993 - 2018

Look Back to Code Forward

Visual Studio Live! (VSLive!™) is thrilled to be returning to Chicago where developers, software architects, engineers and designers will “look back to code forward” during four days of unbiased and cutting-edge education on the Microsoft Platform.

Tackle training on the hottest topics (like .NET Core, Angular, VS2017), debate with industry and Microsoft insiders (people like Rockford Lhotka, Deborah Kurata and Brock Allen) and network with your peers—plus, help us celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Come experience the education, knowledge-share and networking at #VSLive25.

DEVELOPMENT TOPICS INCLUDE:



**DevOps in the
Spotlight**



**Cloud, Containers
and Microservices**



**AI, Data and
Machine Learning**



**Developing New
Experiences**



**Delivery and
Deployment**



**.NET Core
and More**



**Full Stack
Web Development**



vslive.com/chicago

**Register Now
to Save \$400!**

Use Promo Code MSDN

SILVER SPONSOR



SUPPORTED BY



Microsoft



Visual Studio



msdn
magazine

Visual Studio
MAGAZINE

PRODUCED BY



IIO5 MEDIA
YOUR GROWTH. OUR BUSINESS.

DevOps in the Spotlight		Cloud, Containers and Microservices		AI, Data and Machine Learning		Developing New Experiences		Delivery and Deployment		.NET and More		Full Stack Web Development	
START TIME	END TIME	Pre-Conference Workshops: Monday, September 17, 2018 <small>(Separate entry fee required)</small>											
8:00 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries											
9:00 AM	6:00 PM	M01 Workshop: Build a Modern ASP.NET App in the Cloud with a full CI/CD Pipeline in VSTS - <i>Brian Randall</i>				M02 Workshop: SQL Server for Developers - <i>Andrew Brust and Leonard Lobel</i>				M03 Workshop: Distributed Cross-Platform Application Architecture - <i>Rockford Lhotka and Jason Bock</i>			
6:45 PM	9:00 PM	Dine-A-Round											
START TIME	END TIME	Day 1: Tuesday, September 18, 2018											
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries											
8:00 AM	9:15 AM	T01 The State of XAML: Recent Changes for UWP, WPF, Xamarin - <i>Billy Hollis</i>			T02 An Introduction to TypeScript - <i>Jason Bock</i>			T03 SQL Server Security Features for Developers - <i>Leonard Lobel</i>			T04 Get Started with Git - <i>Robert Green</i>		
9:30 AM	10:45 AM	T05 Building Your First Mobile App with Xamarin Forms - <i>Robert Green</i>			T06 Essential Web Development with ASP.NET Core - <i>Mark Michaelis</i>			T07 Exploring T-SQL Enhancements: Windowing and More - <i>Leonard Lobel</i>			T08 DevOps for the SQL Server Database - <i>Brian Randall</i>		
11:00 AM	12:00 PM	KEYNOTE: To Be Announced - <i>Amanda Silver, Partner Director of Program Management, Microsoft</i>											
12:00 PM	1:30 PM	Lunch											
1:30 PM	2:45 PM	T09 Cross-Platform App Dev with C# and CSLA .NET - <i>Rockford Lhotka</i>			T10 Assembling the Web—A Tour of WebAssembly - <i>Jason Bock</i>			T11 Glue for the Internet: Introducing Azure Event Grid - <i>Jeremy Likness</i>			T12 Azure DevOps with VSTS, Docker, and K8 - <i>Brian Randall</i>		
3:00 PM	4:15 PM	T13 A Dozen Ways to Mess Up Your Transition From Windows Forms to XAML - <i>Billy Hollis</i>			T14 Entity Framework Core 2 For Mere Mortals - <i>Philip Japikse</i>			T15 Code First in the Cloud: Serverless .NET with Azure - <i>Jeremy Likness</i>			T16 Essential C# 8.0 - <i>Mark Michaelis</i>		
4:15 PM	5:30 PM	Welcome Reception											
START TIME	END TIME	Day 2: Wednesday, September 19, 2018											
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries											
8:00 AM	9:15 AM	W01 Electron: Desktop Development For Web Developers - <i>Chris Woodruff</i>			W02 JavaScript for the C# (and Java) Developer - <i>Philip Japikse</i>			W03 Quantum Computing and the Future of Software Development - <i>Jerry Nixon</i>			W04 Building a Stronger Team, One Strength at a Time - <i>Angela Dugan</i>		
9:30 AM	10:45 AM	W05 Enhancing UWP Experiences with Fluent Design - <i>Tony Champion</i>			W06 Architecting and Developing Microservices Apps - <i>Eric D. Boyd</i>			W07 Sharing C# Code Across Platforms - <i>Rockford Lhotka</i>			W08 How do You Measure up? Collect the Right Metrics for the Right Reasons - <i>Angela Dugan</i>		
11:00 AM	12:00 PM	General Session: To Be Announced											
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch											
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)											
1:30 PM	2:45 PM	W09 Learning The Language Of HTTP For A Better Data Experience In Your Mobile Apps - <i>Chris Woodruff</i>			W10 Angular 101 - <i>Deborah Kurata</i>			W11 Power BI: What Have You Done for Me Lately? - <i>Andrew Brust</i>			W12 Fault Driven Development - <i>Josh Garverick</i>		
3:00 PM	4:15 PM	W13 Building Cross Device Experiences with Project Rome - <i>Tony Champion</i>			W14 N Things You Didn't Know About the Router - <i>Deborah Kurata</i>			W15 Analytics and AI with Azure Databricks - <i>Andrew Brust</i>			W16 Core Azure Solutions: Automation - <i>Josh Garverick</i>		
4:30 PM	5:45 PM	W17 Use UWP to Modernize Your Existing WinForms and WPF Applications - <i>Walt Ritscher</i>			W18 Tools for Modern Web Development - <i>Ben Hoelting</i>			W19 Create Intelligent Bots with Cognitive Services and Azure Search - <i>Eric D. Boyd</i>			W20 Real World Scrum with Team Foundation Server & Visual Studio Team Services - <i>Benjamin Day</i>		
6:30 PM	9:00 PM	VSLive's Windy City Sunset Cruise											
START TIME	END TIME	Day 3: Thursday, September 20, 2018											
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries											
8:00 AM	9:15 AM	TH01 PowerShell for Developers - <i>Brian Randall</i>			TH02 Docker for ASP.NET Core Developers - <i>Michele Leroux Bustamante</i>			TH03 MVVM and ASP.NET Core Razor Pages - <i>Ben Hoelting</i>			TH04 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - <i>Benjamin Day</i>		
9:30 AM	10:45 AM	TH05 From Waterfall to Agile. Microsoft's Not-So-Easy Evolution into the World of DevOps - <i>Abel Wang</i>			TH06 Developing Microservices Solutions on Azure - <i>Michele Leroux Bustamante</i>			TH07 Eliminate Code Using Data Binding in WPF - <i>Paul Sheriff</i>			TH08 C# 7, Roslyn and You - <i>Jim Wooley</i>		
11:00 AM	12:15 PM	TH09 Writing Testable Code and Resolving Dependencies—DI Kills Two Birds with One Stone - <i>Miguel Castro</i>			TH10 Effective Data Visualization - <i>David Giard</i>			TH11 Store Data Locally for Offline Web Applications - <i>Paul Sheriff</i>			TH12 Improving Code Quality with Static Analyzers - <i>Jim Wooley</i>		
12:15 PM	1:15 PM	Lunch											
1:15 PM	2:30 PM	TH13 Exposing an Extensibility API for Your Applications and Services - <i>Miguel Castro</i>			TH14 Adding Image and Voice Intelligence to Your Apps with Microsoft Cognitive Services - <i>David Giard</i>			TH15 Modern Security Architecture for ASP.NET Core - <i>Brock Allen</i>			TH16 SQL Server 2017—Intelligence Built-in - <i>Scott Klein</i>		
2:45 PM	4:00 PM	TH17 Advanced DevOps—Deep Dive into Feature Flags - <i>Abel Wang</i>			TH18 Programming with Microsoft Flow - <i>Walt Ritscher</i>			TH19 Implementing Authorization in Web Applications and APIs - <i>Brock Allen</i>			TH20 Databases and Data Lakes—Bridging the Gap - <i>Scott Klein</i>		

Speakers and sessions subject to change

CONNECT WITH US



twitter.com/vslive – @VSLive



facebook.com – Search "VSLive"



linkedin.com – Join the "Visual Studio Live" group!

vslive.com/chicago



Introduction to DNN Image Classification Using CNTK

Image classification involves determining what category an input image belongs to, for example identifying a photograph as one containing “apples” or “oranges” or “bananas.” The two most common approaches for image classification are using a standard deep neural network (DNN) or using a convolutional neural network (CNN). In this article I’ll explain the DNN approach, using the CNTK library.

Take a look at **Figure 1** to see where this article is headed. The demo program creates an image classification model for a subset of the Modified National Institute of Standards and Technology (MNIST) dataset. The demo training dataset consists of 1,000 images of handwritten digits. Each image is 28 high by 28 pixels wide (784 pixels) and represents a digit, 0 through 9.

The demo program creates a standard neural network with 784 input nodes (one for each pixel), two hidden processing layers (each with 400 nodes) and 10 output nodes (one for each possible digit). The model is trained using 10,000 iterations. The loss (also known as training error) slowly decreases and the prediction accuracy slowly increases, indicating training is working.

After training completes, the demo applies the trained model to a test dataset of 100 items. The model’s accuracy is 84.00 percent, so 84 of the 100 test images were correctly classified.

This article assumes you have intermediate or better programming skill with a C-family language, but doesn’t assume you know much about CNTK or neural networks. The demo is coded using Python, but even if you don’t know Python, you should be able to follow along without too much difficulty. The code for the demo program is presented in its entirety in this article. The two data files used are available in the download that accompanies this article.

Understanding the Data

The full MNIST dataset consists of 60,000 images for training and 10,000 images for testing. Somewhat unusually, the training set is contained in two files, one that holds all the pixel values and one that holds the associated label values (0 through 9). The test images are also contained in two files.

Additionally, the four source files are stored in a proprietary binary format. When working with deep neural networks, getting the data into a usable form is almost always time-consuming and difficult. **Figure 2** shows the contents of the first training image. The key point is that

each image has 784 pixels, and each pixel is a value between 00h (0 decimal) and FFh (255 decimal).

Before writing the demo program, I wrote a utility program to read the binary source files and write a subset of their contents to text files that can be easily consumed by a CNTK reader object. File `mnist_train_1000_cntk.txt` looks like:

```
|digit 0 0 0 0 0 1 0 0 0 0 |pixels 0 .. 170 52 .. 0
|digit 0 1 0 0 0 0 0 0 0 0 |pixels 0 .. 254 66 .. 0
etc.
```

Getting the raw MNIST binary data into CNTK format isn’t trivial. The source code for my utility program can be found at: bit.ly/2ErcCbW.

There are 1,000 lines of data and each represents one image. The tags “|digit” and “|pixels” indicate the start of the value-to-predict and the predictor values. The digit label is one-hot encoded where the position of the 1 bit indicates the digit. Therefore, in the preceding code, the first two images represent a “5” and a “1.” Each line of data has 784 pixel values, each of which is between 0 and

```
C:\WINDOWS\system32\cmd.exe

C:\MNIST\DNN>python mnist_dnn.py

Begin MNIST classification using a DNN

Creating a 784-(400-400)-10 ReLU classifier
Selected CPU as the process wide default device.

Starting training

batch    0: mean loss = 2.3026, accuracy = 8.00%
batch 1000: mean loss = 2.3040, accuracy = 4.00%
batch 2000: mean loss = 2.2696, accuracy = 30.00%
batch 3000: mean loss = 1.5241, accuracy = 50.00%
batch 4000: mean loss = 0.6649, accuracy = 82.00%
batch 5000: mean loss = 0.5817, accuracy = 86.00%
batch 6000: mean loss = 0.2918, accuracy = 92.00%
batch 7000: mean loss = 0.1840, accuracy = 96.00%
batch 8000: mean loss = 0.1304, accuracy = 98.00%
batch 9000: mean loss = 0.1717, accuracy = 94.00%

Training complete

Model accuracy on the 100 test items = 84.00%

End MNIST classification using a DNN

C:\MNIST\DNN>
```

Figure 1 Image Classification Using a DNN with CNTK

Code download available at msdn.com/magazine/0718magcode.

255. File `mnist_test_100_cntk.txt` has 100 images and uses the same CNTK-friendly format.

In most neural network problems, you want to normalize the predictor values. Instead of directly normalizing the pixel values in the data files, the demo program normalizes the data on the fly, as you'll see shortly.

The Demo Program

The complete demo program, with a few minor edits to save space, is presented in **Figure 3**. All normal error checking has been removed. I indent with two space characters instead of the usual four to save space. Note that the `\` character is used by Python for line continuation.

The `mnist_dnn.py` demo has one helper function, `create_reader`.

All control logic is in the single main function. Because CNTK is young and under continuous development, it's a good idea to add a comment detailing which version is being used (2.4 in this case).

Installing CNTK can be a bit tricky if you're new to the Python world. First you install an Anaconda distribution of Python, which contains the required Python interpreter, the necessary packages such as NumPy and SciPy, and useful utilities such as pip. I used Anaconda3 4.1.1 64-bit, which includes Python 3.5. After installing Anaconda, you install CNTK as a Python package, not a stand-alone system, using the pip utility. From an ordinary shell, the command I used was:

```
>pip install https://cntk.ai/PythonWheel/CPU-Only/cntk-2.4-cp35-cp35m-win_amd64.whl
```

Note the "cp35" in the wheel file that indicates the file is for use with Python 3.5. Be careful; almost all the CNTK installation failures I've seen have been due to Anaconda-CNTK version incompatibilities.

It's usually a good idea to explicitly set the CNTK global random number seed so your results will be reproducible.

The signature of the reader function is `create_reader(path, input_dim, output_dim, rnd_order, m_swps)`. The `path` parameter points to a training or test file that's in CNTK format. The `rnd_order` parameter is a Boolean flag that will be set to True for training data because you want to process training data in random order to prevent oscillating without making training progress. The parameter will be set to False when reading test data to evaluate model accuracy because order isn't important then. The `m_swps` parameter ("maximum sweeps") will be set to the constant

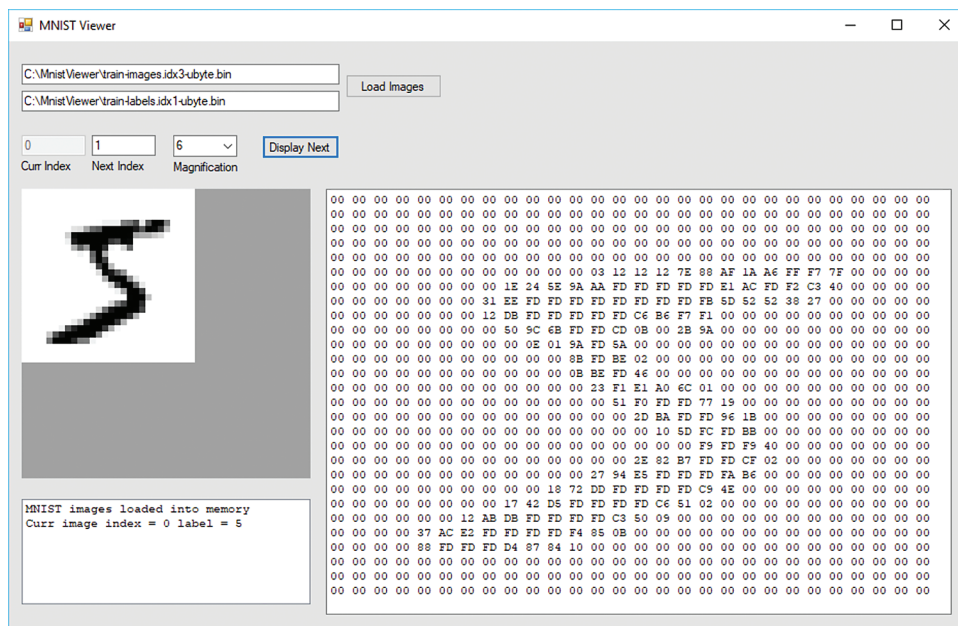


Figure 2 An MNIST Image

INFINITELY_REPEAT for training data (so it can be processed repeatedly) and set to 1 for test data evaluation.

Creating the Model

The demo prepares a deep neural network with:

```
train_file = ".\\Data\\mnist_train_1000_cntk.txt"
test_file = ".\\Data\\mnist_test_100_cntk.txt"
C.cntk_py.set_fixed_random_seed(1)
input_dim = 784
hidden_dim = 400
output_dim = 10
X = C.ops.input_variable(input_dim, dtype=np.float32)
Y = C.ops.input_variable(output_dim) # 32 is default
```

It's usually a good idea to explicitly set the CNTK global random number seed so your results will be reproducible. The number of input and output nodes is determined by your data, but the number of hidden processing nodes is a free parameter and must be determined by trial and error. Using 32-bit variables is the default for CNTK and is typical for neural networks because the precision gained by using 64 bits isn't worth the performance penalty incurred.

The network is created like so:

```
with C.layers.default_options(init=
    C.initializer.uniform(scale=0.01)):
    h_layer1 = C.layers.Dense(hidden_dim,
        activation=C.ops.relu, name='hidLayer1')(X/255)
    h_layer2 = C.layers.Dense(hidden_dim,
        activation=C.ops.relu, name='hidLayer2')(h_layer1)
    o_layer = C.layers.Dense(output_dim, activation=None,
        name='outLayer')(h_layer2)
    dnn = o_layer # train this
    model = C.ops.softmax(dnn) # use for prediction
```

The Python with statement is a syntactic shortcut to apply a set of common arguments to multiple functions. Here it's used to initialize all network weights to random values between -0.01 and +0.01. The X object holds the 784 input values for an image. Notice that each value is normalized by dividing by 255 so the actual input values will be in the range [0.0, 1.0].

The normalized input values act as input to the first hidden layer. The outputs of the first hidden layer act as inputs to the



Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

October 7-11, 2018
Hilton Resort

San Diego

Visual Studio **LIVE!** **25**
YEARS OF CODING INNOVATION



Code With Us in Sunny San Diego!

For the **FIRST TIME EVER** in our 25 year history, Visual Studio Live! is heading to San Diego, CA for up to 5 days of practical, unbiased, Developer training, including **NEW** intense hands-on labs.

Join us as we dig into the latest features of Visual Studio 2017, ASP.NET Core, Angular, Xamarin, UWP and more. Code with industry experts, get practical answers to your current challenges, and immerse yourself in the Microsoft platform. Plus, help us celebrate 25 years of coding innovation and experience the education, knowledge-share and networking at #VSLive25.



BEGIN WITH IN-DEPTH TRAINING, END WITH EXCITEMENT

SILVER SPONSOR



SUPPORTED BY



Microsoft



Visual Studio

msdn
magazine

Visual Studio
MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:



DevOps in the Spotlight



Cloud, Containers
and Microservices



AI, Data and Machine
Learning



Developing New
Experiences



Delivery and Deployment



.NET Core and More



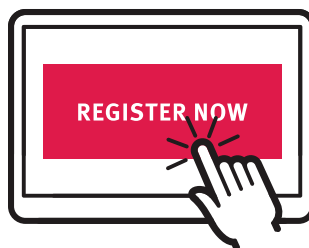
Full Stack Web
Development



Hands-On Labs

Who Should Attend and Why

We've been around since 1993. What's our secret? YOU! Since our first conference (VBITS/VSLive!/Visual Studio Live!) in 1993, tens of thousands of developers, software architects, programmers, engineers, designers and more have trusted us year-in-and-year-out for unbiased and cutting-edge education on the Microsoft Platform.



vslive.com/sandiego

REGISTER TO JOIN US TODAY

Save \$300 When You Register by August 3!

Use Promo Code MSDN

CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

vslive.com/sandiego

second hidden layer. Then, the outputs of the second hidden layer are sent to the output layer. The two hidden layers use ReLU (rectified linear units) activation, which, for image classification, often works better than standard tanh activation.

Notice that there's no activation applied to the output nodes. This is a quirk of CNTK because the CNTK training function expects raw, un-activated values. The `dnn` object is just a convenience alias. The model object has softmax activation so it can be used after training to make predictions. Because Python assigns by reference, training the `dnn` object also trains the model object.

Training the Neural Network

The neural network is prepared for training with:

```
tr_loss = C.cross_entropy_with_softmax(dnn, Y)
tr_error = C.classification_error(dnn, Y)
max_iter = 10000
batch_size = 50
learn_rate = 0.01
learner = C.sgd(dnn.parameters, learn_rate)
trainer = C.Trainer(dnn, (tr_loss, tr_error), [learner])
```

The training loss (`tr_loss`) object tells CNTK how to measure error when training. The cross-entropy error is usually the best

choice for classification problems. The training classification error (`tr_error`) object can be used to automatically compute the percentage of incorrect predictions during training or after training. Specifying a loss function is required, but specifying a classification error function is optional.

The values for the maximum number of training iterations, the number of items in a batch to train at a time, and the learning rate are all free parameters that must be determined by trial and error. You can think of the learner object as an algorithm, and the trainer object as the object that uses the learner to find good values for the neural network's weights and biases values. The stochastic gradient descent (sgd) learner is the most primitive algorithm but works well for simple problems. Alternatives include adaptive moment estimation (adam) and root mean square propagation (rmsprop).

A reader object for the training data is created with these statements:

```
rdr = create_reader(train_file, input_dim, output_dim,
    rnd_order=True, m_swps=C.io.INFINITELY_REPEAT)
mnist_input_map = {
    X : rdr.streams.x_src,
    Y : rdr.streams.y_src
}
```

Figure 3 Complete Demo Program Listing

```
# mnist_dnn.py
# MNIST using a 2-hidden layer DNN (not a CNN)
# Anaconda 4.1.1 (Python 3.5.2), CNTK 2.4

import numpy as np
import cntk as C

def create_reader(path, input_dim, output_dim, rnd_order, m_swps):
    x_strm = C.io.StreamDef(field='pixels', shape=input_dim,
        is_sparse=False)
    y_strm = C.io.StreamDef(field='digit', shape=output_dim,
        is_sparse=False)
    streams = C.io.StreamDefs(x_src=x_strm, y_src=y_strm)
    deserial = C.io.CTFDeserializer(path, streams)
    mb_src = C.io.MinibatchSource(deserial, randomize=rnd_order,
        max_sweeps=m_swps)
    return mb_src

# =====
def main():
    print("\nBegin MNIST classification using a DNN\n")

    train_file = ".\\Data\\mnist_train_1000_cntk.txt"
    test_file = ".\\Data\\mnist_test_100_cntk.txt"

    C.cntk_py.set_fixed_random_seed(1)
    input_dim = 784 # 28 x 28 pixels
    hidden_dim = 400
    output_dim = 10 # 0 to 9

    X = C.ops.input_variable(input_dim, dtype=np.float32)
    Y = C.ops.input_variable(output_dim) # float32 is default

    print("Creating a 784-(400-400)-10 ReLU classifier")
    with C.layers.default_options(init=\\
        C.initializer.uniform(scale=0.01)):
        h_layer1 = C.layers.Dense(hidden_dim, activation=C.ops.relu,
            name='hidLayer1')(X/255)
        h_layer2 = C.layers.Dense(hidden_dim, activation=C.ops.relu,
            name='hidLayer2')(h_layer1)
        o_layer = C.layers.Dense(output_dim, activation=None,
            name='outLayer')(h_layer2)
    dnn = o_layer # train this
    model = C.ops.softmax(dnn) # use for prediction

    tr_loss = C.cross_entropy_with_softmax(dnn, Y)
    tr_error = C.classification_error(dnn, Y)

    max_iter = 10000 # num batches, not epochs
    batch_size = 50
    learn_rate = 0.01
    learner = C.sgd(dnn.parameters, learn_rate)
    trainer = C.Trainer(dnn, (tr_loss, tr_error), [learner])

    # 3. create reader for train data
    rdr = create_reader(train_file, input_dim, output_dim,
        rnd_order=True, m_swps=C.io.INFINITELY_REPEAT)
    mnist_input_map = {
        X : rdr.streams.x_src,
        Y : rdr.streams.y_src
    }

    # 4. train
    print("\nStarting training\n")
    for i in range(0, max_iter):
        curr_batch = rdr.next_minibatch(batch_size, \\
            input_map=mnist_input_map)
        trainer.train_minibatch(curr_batch)
        if i % int(max_iter/10) == 0:
            mcee = trainer.previous_minibatch_loss_average
            macc = (1.0 - trainer.previous_minibatch_evaluation_average) \\
                * 100
            print("batch %4d: mean loss = %0.4f, accuracy = %0.2f%% " \\
                % (i, mcee, macc))
    print("\nTraining complete\n")

    # 5. evaluate model on test data
    rdr = create_reader(test_file, input_dim, output_dim,
        rnd_order=False, m_swps=1)
    mnist_input_map = {
        X : rdr.streams.x_src,
        Y : rdr.streams.y_src
    }

    num_test = 100
    test_mb = rdr.next_minibatch(num_test, input_map=mnist_input_map)
    test_acc = (1.0 - trainer.test_minibatch(test_mb)) * 100
    print("Model accuracy on the %d test items = %0.2f%% " \\
        % (num_test, test_acc))

    print("\nEnd MNIST classification using a DNN\n")

if __name__ == "__main__":
    main()
```

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE

Download a Free Trial at

<https://downloads.groupdocs.com>

Microsoft
.NET



 GROUPDOCS



GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.



GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.



GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.



GroupDocs.Comparison

Compare two documents and get a difference summary report.



GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.



GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.



GroupDocs.Metadata

Organize documents with metadata within any cross platform application.



GroupDocs.Search

Transform your document search process for advance full text search capability.

► GroupDocs.Text

► GroupDocs.Editor

► GroupDocs.Parser

► GroupDocs.Watermark

Americas: +1 903 306 1676

EMEA: +44 141 628 8900
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

Figure 4 Training

```
print("\nStarting training \n")
for i in range(0, max_iter):
    curr_batch = rdr.next_minibatch(batch_size, \
        input_map=mnist_input_map)
    trainer.train_minibatch(curr_batch)
    if i % int(max_iter/10) == 0:
        mcee = trainer.previous_minibatch_loss_average
        macc = (1.0 - \
            trainer.previous_minibatch_evaluation_average) \
            * 100
        print("batch %4d: mean loss = %0.4f, accuracy = \
            %0.2f%%" % (i, mcee, macc))
```

If you examine the `create_reader` code in **Figure 3**, you'll see that it specifies the tag names ("pixels" and "digit") used in the data file. You can consider `create_reader` and the code to create a reader object as boilerplate code for DNN image classification problems. All you have to change is the tag names, and the name of the mapping dictionary (`mnist_input_map`).

After everything is prepared, training is performed, as shown in **Figure 4**.

The demo program is designed so that each iteration processes one batch of training items. Many neural network libraries use the term "epoch" to refer to one pass through all training items. In this example, because there are 1,000 training items, and the batch size is set to 50, one epoch would be 20 iterations.

An alternative to training with a fixed number of iterations is to stop training when loss/error drops below some threshold. It's important to display loss/error during training because training failure is the rule rather than the exception. Cross-entropy error is difficult to interpret directly, but you want to see values that tend to get smaller. Instead of displaying average classification error ("25 percent wrong"), the demo computes and prints the average classification accuracy ("75 percent correct"), which is a more natural metric in my opinion.

Evaluating and Using the Model

After an image classifier has been trained, you'll usually want to evaluate the trained model on test data that has been held out. The demo computes classification accuracy as shown in **Figure 5**.

A new data reader is created. Notice that unlike the reader used for training, the new reader doesn't traverse the data in random order, and that the number of sweeps is set to 1. The `mnist_input_map` dictionary object is recreated. A common mistake is to try and use the original reader—but the `rdr` object has changed so you need to recreate the mapping. The `test_minibatch` function returns the average classification error for its mini-batch argument, which in this case is the entire 100-item test set.

Figure 5 Computing Classification Accuracy

```
rdr = create_reader(test_file, input_dim, output_dim,
    rnd_order=False, m_swps=1)
mnist_input_map = {
    X : rdr.streams.x_src,
    Y : rdr.streams.y_src
}
num_test = 100
test_mb = rdr.next_minibatch(num_test,
    input_map=mnist_input_map)
test_acc = (1.0 - trainer.test_minibatch(test_mb)) * 100
print("Model accuracy on the %d test items = %0.2f%%" \
    % (num_test, test_acc))
```

After training, or during training, you'll usually want to save the model. In CNTK, saving would look like:

```
mdl_name = "..\\Models\\mnist_dnn.model"
model.save(mdl_name)
```

This would save using the default CNTK v2 format. An alternative is to use the Open Neural Network Exchange (ONNX) format. Notice that you'll generally want to save the model object (with softmax activation) rather than the `dnn` object (no output activation). From a different program, a saved model could be loaded into memory along the lines of:

```
mdl_name = "..\\Models\\mnist_dnn.model"
model = C.ops.functions.Function.load(mdl_name)
```

After loading, the model can be used as if it had just been trained. The demo program doesn't use the trained model to make a prediction. Prediction code could resemble this:

```
input_list = [0.55] * 784 # [0.55, 0.55, . . . 0.55]
input_vec = np.array(input_list, dtype=np.float32)
pred_probs = model.eval(input_vec)
pred_digit = np.argmax(pred_probs)
print(pred_digit)
```

Many neural network libraries use the term "epoch" to refer to one pass through all training items.

The `input_list` has a dummy input of 784 pixel values, each with value 0.55 (recall the model was trained on normalized data so you must feed in normalized data). The pixel values are copied into a NumPy array. The call to the `eval` function would return an array of 10 values that sum to 1.0 and can loosely be interpreted as probabilities. The `argmax` function returns the index (0 through 9) of the largest value, which is conveniently the same as the predicted digit. Neat!

Wrapping Up

Using a deep neural network used to be the most common approach for simple image classification. However, DNNs have at least two key limitations. First, DNNs don't scale well to images that have a huge number of pixels. Second, DNNs don't explicitly take into account the geometry of image pixels. For example, in an MNIST image, a pixel that's directly below a second pixel is 28 positions away from first pixel in the input file.

Because of these limitations, and for other reasons, too, the use of a convolutional neural network (CNN) is now more common for image classification. That said, for simple image classification tasks, using a DNN is easier and often just as (or even more) effective than using a CNN. ■

Dr. James McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.

THANKS to the following Microsoft technical experts who reviewed this article: Chris Lee, Ricky Loynd, Ken Tran

Spreadsheets Everywhere.



SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.



Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.



Windows
Forms



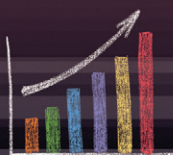
Silverlight



WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.



Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.



Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Download your free fully functional evaluation at SpreadsheetGear.com



SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

JOIN US AT
The Ultimate Education Destination

2018 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
DECEMBER 2-7, 2018



Live! 360: A Unique Conference for the IT and Developer Community

- 6 FULL Days of Training Including Hands-On Labs & Workshops
- 6 Co-Located Conferences for 1 Low Price
- Customize Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360



twitter.com/live360
@live360



facebook.com
Search "Live 360"



linkedin.com
Join the "Live! 360" group!



instagram.com
@live360_events

EVENT PARTNERS



SILVER SPONSOR



SUPPORTED BY



HANDS-ON LABS



Join us for a full-day of pre-conference Hand-On Labs on Sunday, December 2.



**SUMMER SAVINGS =
BEST SAVINGS!**

**REGISTER
NOW**

**REGISTER BY 8/31
AND SAVE \$500!**

Use promo code: MSDN

See website for details.

6 CO-LOCATED CONFERENCES, 1 GREAT PRICE!

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live! features unbiased and practical development training on the Microsoft Platform. Come join us and code in paradise!

SQL Server **LIVE!**
TRAINING FOR DBAs AND IT PROS

SQL Server Live! will leave you with the skills needed to drive your data to succeed, whether you are a DBA, developer, IT Pro, or Analyst.

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

TechMentor is where IT training meets sunshine, with zero marketing speak on topics you need training on now, and solid coverage on what's around the corner.

**NEW!
IN 2018** **Artificial
Intelligence** **LIVE!**
AI FOR DEVELOPERS AND DATA SCIENTISTS

Artificial Intelligence Live! is an innovative, new conference for current and aspiring developers, data scientists, and data engineers covering artificial intelligence (AI), machine learning, data science, Big Data analytics, IoT & streaming analytics, bots, and more.

**Office &
SharePoint** **LIVE!**
ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live! provides leading-edge knowledge and training to work through your most pressing projects and enable people to work from anywhere at any time.

ModernApps **LIVE!**
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!, presented in partnership with Magenic, focuses on how to architect, design and build a complete Modern App from start to finish.



LIVE360EVENTS.COM



Building Better Meetings

I've had some time to digest events at the Build 2018 conference. Here's what impressed me most, as Microsoft continues its transition into an artificial intelligence (AI) company.

The best demo, as usual, involved Scott Guthrie. To demonstrate the Intelligent Edge of the Cloud, a speaker trained Microsoft's AI photo recognition service with images of ScottGu. He then downloaded and ran that model on a Raspberry Pi. The tiny, cheap computer with its built-in camera could now recognize the real ScottGu onstage, red shirt and all. That drew cheers, some of them mine.

The best forward-looking idea came when a speaker demonstrated voice commands, saying: "Hey, Cortana, set up a meeting right now with the smart building team," and the AI voice came back, "Sure thing, you are all free now." Then the speaker said, "Find me a conference room with a Surface Hub," and by golly, one just happened to be open. The meeting was duly scheduled, and the participants notified.

The attendees around me rolled their eyes—what are the odds that the entire team *and* a room would be free on such short notice? But I think Microsoft has stumbled onto something immensely valuable. The company simply has to re-orient the application of their AI to meetings.

We don't need or want meetings made easier to schedule. We already have way too many. ("Where the minutes are kept and the hours are lost," right?) We need fewer meetings, with fewer attendees and a tighter focus. Imagine Microsoft Meeting Blaster™, a new skill for Cortana.

We don't need or want meetings
made easier to schedule. We
already have way too many.

Suppose whenever anyone tried to schedule a meeting, Cortana would cross-examine him about the agenda: "Are you sure you need a meeting on that topic? Tell me the goals you hope to accomplish. Did you see what so-and-so just published about this topic? There, I've sent you the link. Read that first, then come back if you still think you need a meeting." The scheduler would have to do his homework before Cortana would let him distract busy people.

Another huge problem: Because the size of a meeting indicates a leader's importance, managers invite way too many people for

them all to be productive. Invariably you have two or three guys talking and nine guys checking their phones and rolling their eyes, praying for Scotty to beam them out of there. Suppose Microsoft's AI could monitor a meeting via cameras and microphones, producing hard data on that meeting's value for each attendee. Maybe that ammunition could help Cortana resist the encroachment of meetings on their productive time.

Cortana could insist that the scheduler justify each attendee: "You're inviting Bob? The last time Bob came to a meeting on this topic, he spent 30 seconds talking, 150 seconds actually listening, and the remaining 97 minutes yawning and playing Solitaire. And I see they're serving his favorite three-bean chili for lunch that day, and the windows in the room I reserved don't open. I'd skip him if I were you." Or, better yet: "The last time Alice got dragooned into a meeting on this topic, approximately 45.3 seconds contained content that was valuable to her. Convince me this meeting will be different before I let you bother her."

Once a meeting starts, the biggest problem is keeping participants on topic. Think of all the self-indulgent storytelling and tangential screeds you've had to sit through while your deadlines ticked away. Suppose Cortana could listen in and drop the hammer on meeting hogs: "[Referee's whistle] Hey, Charlie. Back on topic, please." "But Cortana, I was just ..." "Right now, please. Don't make me bust you back to Level Zero on Candy Crush." Because Cortana doesn't depend on the leader's evaluation to keep her job, she could slap down even senior miscreants.

How could you train such an AI model? Easy: Unleash Microsoft's AI to do what it claims it can do—recognize facial expressions and body language. Smiles, nodding, thumbs-up gestures (automatically adjusted for differing cultures, of course)—all good. Eye rolls, yawns, silently mouthing "[Expletive] this [expletive]" (in whatever language), taps and drags on phone screens indicating Solitaire, outright snoring—not so good. Now *that's* using AI to benefit society. *That* would make Microsoft a boatload of money and guarantee a standing ovation at Build 2019. I'm looking forward to seeing ScottGu demonstrate it. ■

DAVID S. PLATT teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.

Learn, Explore, Use

Your Destination for Data Cleansing & Enrichment APIs



Your centralized portal to discover our tools, code snippets and examples.

RAPID APPLICATION DEVELOPMENT

Convenient access to Melissa APIs to solve problems with ease and scalability.

REAL-TIME & BATCH PROCESSING

Ideal for web forms and call center applications, plus batch processing for database cleanup.

TRY OR BUY

Easy payment options to free funds for core business operations.

FLEXIBLE CLOUD APIS

Supports REST, JSON, XML and SOAP for easy integration into your application.

Turn Data into Success – Start Developing Today!

Melissa.com/developer

1-800-MELISSA

melissa



Rider

New .NET IDE

**Cross-platform.
Powerful.
Fast.**

From the makers of ReSharper,
IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider



**JET
BRAINS**